New Datasets and Improved Performance: Predicting Alternative Splicing Behaviour using Deep Learning

1044935

St Cross College

University of Oxford

Submitted in partial completion of the

MSc in Computer Science

Trinity 2020

Contents

Glossary **iv**

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Contribution . . . 1

2 Background **2**

3 Related work **6**

4 Methods **10**
4.1 Task formulation . . . 11
4.2 Datasets . . . 11
4.2.1 HEXEvent database . . . 12
4.2.2 GTEx . . . 12
4.2.3 HipSci . . . 13
4.3 Data processing . . . 14
4.3.1 Estimating PSI . . . 14
4.3.2 Final sample processing . . . 24
4.3.3 Dataset statistics . . . 26
4.4 Models . . . 28
4.4.1 DSC: CNN-based . . . 29
4.4.2 D2V: MLP-based . . . 31
4.4.3 BiLSTM + Attn . . . 36
4.4.4 Alternative implementations of attention . . . 40
4.5 Training and implementation details . . . 45
4.5.1 Implementation . . . 45
4.5.2 Training . . . 46

5 Results **49**
5.1 HEXEvent dataset . . . 49
5.2 GTEx-based datasets . . . 52
5.2.1 Cassette exon-based datasets . . . 52

5.2.2 Junction-based datasets 55
5.2.3 Reconstructing HEXEvent-dataset with GTEx data 56
5.3 HipSci-based datasets 56
5.3.1 SUPPA with neuron tissue induced iPSCs 56
5.3.2 MAJIQ with iPSCs differentiated to neurons (exons) 57
5.3.3 MAJIQ with iPSCs differentiated to neurons (junctions) . . 59
5.3.4 MAJIQ with undifferentiated iPSCs 59

6 Discussion / Conclusion **61**

7 Appendix **62**
7.0.1 Additional Doc2Vec training details 63

References **65**

Glossary

Splicing . . . . Process by which a pre-mRNA is converted to a mature mRNA or the actual act of cutting out genomic sequences in that process itself.

Exon . . . . . Genomic sequence which is typically kept from the pre-mRNA during splicing.

Intron . . . . . Genomic sequence which is typically removed from the pre-mRNA during splicing.

PSI . . . . . Percent-spliced in, in what proportions of transcript a specific exon (or junction) is contained in the mature mRNA.

Motif . . . . . A widespread nucleotide sequence pattern conjectured to be biologically significant.

EST . . . . . Expressed sequence tag, a short cDNA sequence used in older sequencing techniques.

GTEx . . . . . Genotype-Tissue Expression project, they provide a large repository of sequencing data which we use.

iPSC . . . . . induced pluripotent stem cells, mature cells which have been reprogrammed to again become pluripotent (undetermined).

HipSci . . . . . Human Induced Pluripotent Stem Cell Initiative, a repository of iPSC-based data which we use.

DSC . . . . . Deep Splicing Code, a model used for constitutive exon classification.

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

"Why genes in pieces?" – can put this citation at the top and start with related story

1.1 Motivation

1.2 Contribution

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]:

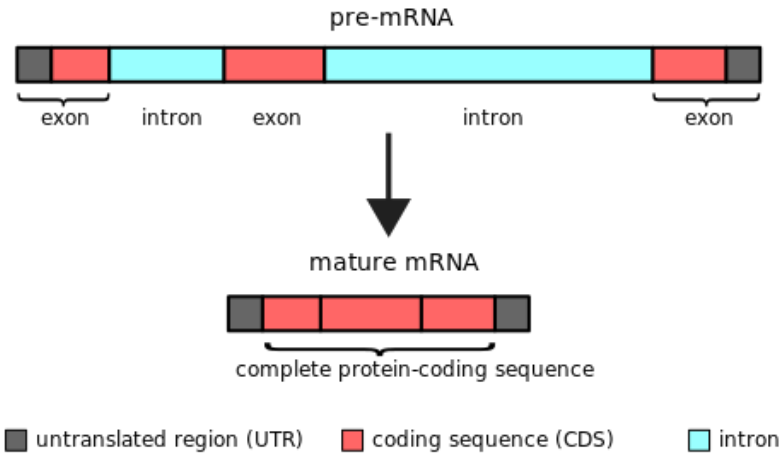

Figure 2.1: The process of splicing [1]. Introns are removed from the pre-mRNA to obtain the mature mRNA only consisting of exons. Apart from coding regions, exons may also consist of non-coding untranslated regions (UTRs). Like introns, UTRs influence gene expression.

- 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

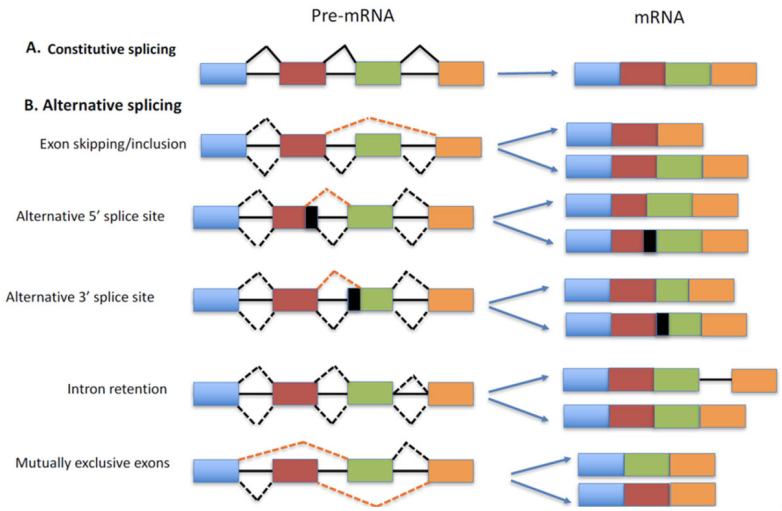

Figure 2.2: Visualization of the most common forms of alternative splicing and the resulting different possible mature mRNAs [5].

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 human protein-coding 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 [2]. 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] [10]. As a result, alternative splicing has also been suggested as a biomarker and potential target for drug discovery [11].

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 [12][13]. 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.

[12] 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: Ψ is defined as the proportion of transcripts out of all transcripts that contain a given exon [14]. In other words, given a random transcript, PSI denotes the probability of a particular exon being included or excluded.

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 [15] [16]. 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. [17] 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 [18]. Using a Deep Neural Network (DNN) with an autoencoder, they were able to improve upon the results achieved by a BNN model. Albeit [18] initially used a different dataset and a different task formulation than [17], [19] were able to show that these improvements also lasted when directly comparing the models on the same dataset using the same task formulation. Furthermore, [19] 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 [18]. 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, [20] 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 [21], 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. [15] For predicting whether an exon is constitutive or not, features around the cassette exon itself have been shown to be the most critical [22].

[23] 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. [24] 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

In this chapter, we give more details about the exact task we are trying to solve, introduce the primary data sources and describe how these were used to obtain the training datasets. Additionally, we motivate and explain the evaluated models and describe how these were implemented and trained.

Why PSI?

- why not more ambitious? why not predict whole transcript directly? –> but this should be somewhere else, like in the task formulation section

ok, older works use PSI – but why do we use it?

High-throughput sequencing methods are a popular and 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 [25]. -> this is why you introduce alternative splicing events; as reads are too short to identify splicing events

4.1 Task formulation

The output of our models will be the classification of a given exon or junction as constitutive or non-constitutive. To obtain the training label of an exon or junction as constitutive or non-constitutive, we need its respective PSI value. Strictly speaking, all exons and junctions with a PSI of less than 100% are non-constitutive. However, due to spurious reads, we classify all exons with a PSI $\leq$ 99% as constitutive.

Previous work indicates that advances in predicting one type of alternative splicing behaviour (e.g. cassette exons) via Machine Learning methods also translates to advances in prediction for other types (e.g. exons with an alternative 3' splice site) [24] [23]. 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 eukaryote and thus we choose cassette exons as the type of alternative splicing we will focus on.

4.2 Datasets

Three primary sources for genomic data were used in this study: the Human Exon splicing events (HEXEvent) database [26], data from the Genotype-Tissue Expression (GTEx) project [27] and data from the Human Induced Pluripotent Stem Cell Initiative (HipSci) [28]. Except for the HEXEvent database, none of these primary sources directly report the PSI values of exons or junctions. However, we will use the PSI value as our prediction target and thus apply further processing steps to obtain PSI estimates 4.3.1. 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 4.3.2.

4.2.1 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). 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. ESTs are short cDNA sequences which have been used in older sequencing techniques (before the advent of modern high-throughput sequencing technologies).

Some issues with ESTs which therefore also affect HEXEvent are worth highlighting. EST are based on only a single sequencing pass and especially bases at the 3' and 5' end of the EST are known to be highly error prone [29]. Additionally, ESTs underpresent less frequent transcripts and as a result often only capture 50 - 65% of an organism's genes [30]. Thus, these biases may also have an adverse effect on the HEXEvent database's data quality.

4.2.2 GTEx

The GTEx project provides the most comprehensive database for tissue-specific gene expression and regulation available to-date; containing over 17000 samples from nearly 1000 human donors. The sequencing data was obtained using mainly molecular assay-based techniques like Whole Genome Genotyping (WGS), Whole Exome Sequencing (WES) and RNA-Seq. Samples were taken from up to 54 different tissue sites. In particular, the tissue samples are also taken from less commonly seen tissue sites such as brain or heart as the GTEx project sources its samples from recently-deceased donors who have donated their body to science. Thus, the GTEx project, and by extension the parts of this thesis relying on GTEx data, was only made possible through the kindness and generosity of donors making their body available for science.

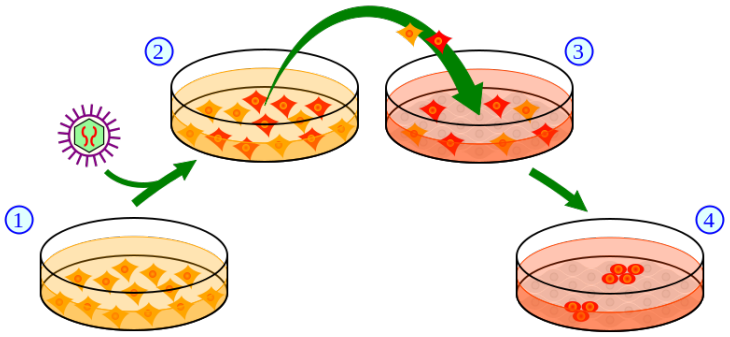

Figure 4.1: The four high-level steps taken to obtain iPSC cells [31]: (1) Grow a cell culture of donor cells, (2) transduce the genes associated with reprogramming into the cells via viral vectors, (3) isolate the cells expressing the transduced genes (in red) and culture them according to embryonic stem cell culture and, finally, (4) a small subset of the transfected cells become iPSC cells.

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

4.2.3 HipSci

HipSci provides a large repository of human induced pluripotent stem cell (iPSC) lines. iPSC cells are mature cells which have been reprogrammed to again become pluripotent (undetermined). This process is visualized in Figure 4.1. As iPSC cell lines are not directly obtained from human donors, but rather from building a cell culture based on some initial donor cells, accessing raw RNA-seq reads is

not constrained by the same data privary regulations which prevented access to the raw RNA-seq reads from the GTEx project.

Concretely, over 300 cell lines from over 300 donors along with sequencing information based on RNA-seq is publicly available from the HipSci portal. From these we selected two different groups of cell lines:

- the first group contains 25 biological replicates (that is, samples from different cell lines developed under the same conditions) of sensory neuron cell lines [32]. These cell lines were obtained by differentiating iPSC cells to sensory neuron cells.
- the second group contains 20 biological replicates of iPSC cell lines which weren't differentiated to any other cell type. Here care was taken to choose donors from whom at least two different iPSC cell lines were developed.

All used iPSC cells were originally obtained from skin cells.

Selecting the samples in this way, enables us to test cross-condition performance in three ways:

- on the same cell type from the same donor, but from a different cell line. Here we expect the best generalization performance.
- on the same cell type, but a different donor.
- on a different cell type and from a different donor. Here we expect the worst generalization performance.

The appendix contains the ENA Accession Numbers of the chosen samples 7.

4.3 Data processing

4.3.1 Estimating PSI

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

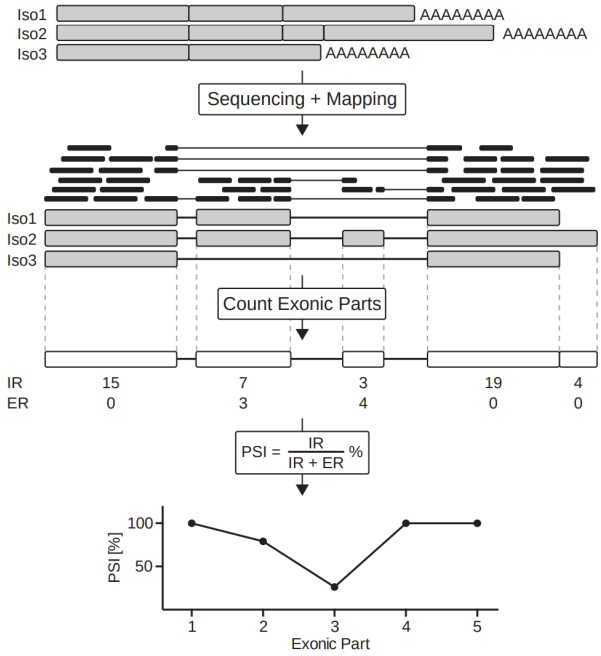

Figure 4.2: Process of estimating PSI based off RNA-seq reads [25]. The reads are mapped to the transcript. Based on annotation files, the reads overlapping exon/intron junctions are classfied as including or excluding reads for a particular exon. Based on the observed IR and ER, the PSI for a given exon is estimated.

exon being excluded. A PSI value of 100% indicates a constitutive exon which 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 also has various issues:

(a) 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 they likely shouldn't.

To account for this, it is possible to only count exons who experienced at least a certain number of reads. However, raw read counts are very unreliable when not accounting for gene expressiveness: 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). Of these, TPM is usually the most common choice. Therefore, a better solution is likely to only count reads from genes whose TPM is above a certain threshold.

Note that there is not a principled way to choose this threshold. The choice between a threshold which e.g. filters 5% or 20% of the samples is a trade-off between data quality and data quantity. In practice, the 'optimal' cutoff threshold will be unknown and vary from dataset to dataset.

(b) 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.

(c) Related to (a), 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

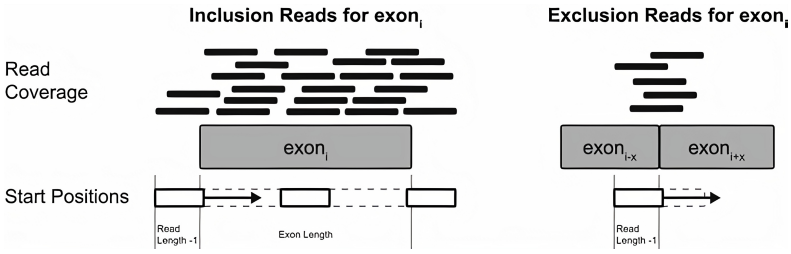

Figure 4.3: This graphic showcases the need for read length normalization when estimating PSI [25]. An IR (left) can be located anywhere over the exon and around the junction. An ER (right) can only be located where it crosses the exon-exon junction. Thus, any PSI estimation which does not account for this will be biased towards estimating higher PSI values than accurate.

this is not obvious. Read depths between multiple libraries may vary and this should be accounted for.

(d) IRs and ERs must be normalized for exon length to obtain meaningful results. For a long exon, the majority of reads will be IRs because they can be located over a much larger area than the ERs who must overlap with the 0-length feature of the splicing junction (see Figure 4.3). This can be accounted for by normalizing for the possible number of start positions of each read population [25]:

$$\#IR_{norm} = \frac{\#IR}{exon\_length + read\_length - 1}$$

$$\#ER_{norm} = \frac{\#ER}{read\_length - 1}$$

(e) Reads may be misattributed. The motivation for analyzing splicing at a level of alternative splicing events is that full gene isoforms can not reliably be quantified given the currently available short RNA-seq read lengths. However, this issue may still occur at the splicing event level when two splicing variants of an exon share a junction. This is illustrated in Figure 4.4. This misattribution of a read can only be corrected by considering circumstantial evidence, similar to (b); for instance, read counts on the exon junction may give evidence regarding to which splicing event the read at the shared junction belongs.

Any estimate of PSI that doesn't account for all, or at least the majority of these issues, will likely be very unreliable.

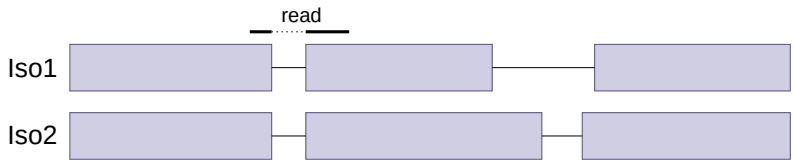

Figure 4.4: The read across the exon-exon junction may be attributed to isoform 1 or isoform 2. Consequently, it may also be attributed to the first or second splicing variant of the middle exon.

A word of caution

However, even if all of these issues are perfectly handled, the resulting PSI estimate would still not be perfect. For instance, the 'solutions' for (a) and (d) implicitly rely on the assumption that reads across a transcript are evenly distributed. It is well-known that sequence reads across a transcript aren't equally distributed due to biases relating to GC-content [33], gene length and dinucleotide frequencies [34]. These biases are not easily quantifiable and can never be perfectly corrected for. Thus, the PSI estimate will always be affected by these biases.
In more general terms, the data on which the estimate is based on is biased in multiple complex, dynamically interacting ways and this will always lead to downstream effects independent of the preprocessing method used. The question is not whether the resulting estimate is still biased, but rather if the effect of systematic biases influencing it can be alleviated enough for the estimate to be useful. In this study, we answer this question in an empirical way by testing datasets resulting from four different PSI estimation methods. We now introduce the exact datasets we use and which PSI estimatin method was used to obtain them.

Dataset based on HEXEvent

The HEXEvent database already provides PSI estimates based on EST counts for each exon and facilitates standard filtering steps (e.g. only select cassette exons). The baseline paper [24] takes HEXEvent as a starting point and generates three different datasets respectively only containing cassette exons, exons with an alternative 3' splice and exons with an alternative 5' splice site. Each of these

datasets also contains constitutive exons. To account for the biases inherent in EST data, it applies multiple filtering steps such as requiring a minimum number of supporting ESTs and only using exons which display a single type of alternative splicing. They make the resulting datasets publicly available and we use the dataset containing cassette exons for direct comparability.

However, it should be noted that HEXEvent uses the critized naive estimation method for PSI, i.e., they estimate PSI as $PSI = \frac{\#IR}{(\#IR+\#ER)}$. Except for requiring a minimum number of supporting ESTs, none of the five main critique points above are explicitly accounted for by [24]. The EST data available by UCSC is based on many different tissues which may lead to the derived alternative splicing behaviour being an average of the splicing behaviour of many tissues which never occurs in any singular tissue. Therefore, care should be taken when drawing conclusions based on the PSI estimates of this dataset.

Our PSI estimation implementation

To obtain a PSI estimate based on the publicly accessible exon-exon junction read counts of the GTEx project, we implement our own PSI estimation method. We are not able to use the PSI estimation methods from the literature we introduce below, as they require access to the raw RNA-seq reads which we don't have access to due to data privacy regulations.

GTEx publicly gives access to anonymized information about which sample types were taken from which donor. We use this information to find a donor from whom brain cortex tissue, cerebellum tissue and heart tissue samples were taken. For each tissue sample we derive two datasets: one exon-centric dataset in which we estimate PSI for cassette exons and one intron-centric dataset in which we estimate PSI for junctions. We obtain six datasets in total.

Fundamentally, our PSI estimation relies on the formula $PSI = \frac{\#IR}{(\#IR+\#ER)}$. However, before computing PSI, we also apply the following filtering and normalization steps:

- Cassette exons which are shorter than 25 nt and introns which are shorter than 80 nt are excluded. Exons and introns of these lengths constitute less than 1% of exons and introns in total and are usually an artifact of sequencing errors [24].
- GTEx provides access to TPM values for each gene from a sample. Adressing point (a) from 4.3.1, we obtain the TPM-adjusted read counts via $\#IR^{TPM} = \frac{\#IR}{TPM}$ and $\#ER^{TPM} = \frac{\#ER}{TPM}$. To address spurious reads from lowly-transcribed genes having a disproportionate impact, we only count reads from genes whose TPM is larger than 10. This filters around 84% of genes which in turn filters around 56% of the junctions. This is the most stringent filtering step we apply.
- Adressing point (d) about IRs naturally being more common than ER, we estimate the read length of the RNA-seq reads to be 150 on average [35] and apply the proposed normalization method: $\#IR^{TPM}_{norm} = \frac{\#IR^{TPM}}{(exonlength+149)}$ and $\#ER^{TPM}_{norm} = \frac{\#ER^{TPM}}{149}$.

Using the normalized #IR and #ER of all exons and junctions remaining after filtering, we then compute PSI: $PSI = \frac{\#IR^{TPM}_{norm}}{(\#IR^{TPM}_{norm}+\#ER^{TPM}_{norm})}$.

Comparing our estimation method to the standards set out by 4.3.1, this estimate fails to take into account the information contained in non-junction reads, as we simply don't have access to them. This estimate also does not aggregate the information from multiple biological samples, although we took care to choose the donor with tissue samples from brain cortex, cerebellum and heart, whose total read count was the highest. Reads may also be misattributed in our cassette exon PSI estimation as described in (e). Note that misattribution is not an issue when estimating the PSI of junctions.

More sophisticated PSI estimation approaches

Several approaches have been developed which try to address the 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 interested in. Of the methods directly reporting PSI, we chose two fast methods, representing different approaches to PSI estimation: SUPPA and MAJIQ. SUPPA is primarily based on quantification of transcripts, while MAJIQ is based on building up a splicing graph.

SUPPA

SUPPA [36] 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 $PSI = \frac{TPM_{IR}}{TPM_{IR}+TPM_{ER}}$. TPM_{IR} and TPM_{ER} respectively refer to the total TPM of transcripts supporting a certain exon being included or excluded. SUPPA can integrate the TPM values from multiple samples.

Using SUPPA: SUPPA 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 [37] was used for quantifying the relative abundance of each transcript in TPM. Salmon's quantification is based on the raw RNA-seq reads in FASTQ format, takes into account experimental attributes and corrects 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), SUPPA estimates the PSI for each alternative splicing event.

SUPPA operates on at a transcript level and therefore implicitly relies on the assumption that all transcripts of a given gene are known - this is often not the case. Of the discussed issues in 4.3.1, it accounts for 2 of them:

SUPPA's main competitive advantage for PSI estimation lies in its speed and low memory overhead compared to other methods [38] (which it achieves through leveraging fast transcript quantification methods such as Salmon), not necessarily that it achieves the best accuracy. This becomes apparent when judging how well SUPPA addresses the issues we laid out in 4.3.1: of these adresses it only (c) - the integration of evidence from multiple samples. SUPPA only focuses on alternatively spliced exons and thus we are not able to derive a list of constitutive exons from SUPPA. Since SUPPA operates at the transcript level, it implicitly relies on the assumption that all transcripts of a given gene are known - this is often not the case. Therefore, it remains to be seen whether the estimation of SUPPA is accurate enough for our purposes.

MAJIQ

Modeling Alternative Junction Inclusion Quantification (MAJIQ) builds up a splice graph [39] 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). This problem formulation allows MAJIQ to model more complex splicing events (rather than the ones from 2.2), although we won't leverage this extra capability here. Figure 4.6 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. The only issue from 4.3.1 MAJIQ doesn't account for is (b): the integration of the information from non-junction reads. Thus, we expect MAJIQ's PSI estimates to be the most accurate of all methods we are testing.

Using MAJIQ: MAJIQ requires sequence files in bam format (along with their respective index files) and an annotation DB (we used human GRCh38 release 13)

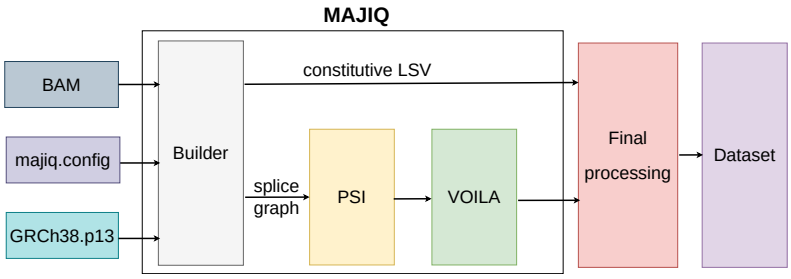

Figure 4.5: Data flow when using MAJIQ to create a training dataset.

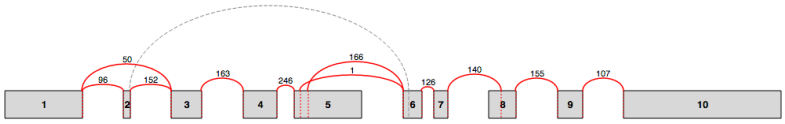

Figure 4.6: An example splice graph as displayed in VOILA [39]. Exons, represented by grey rectangles, are connected with another exon via an edge, if a junction between the exons (or a variant of them) is observed. The number above an edge displays the number of observed raw reads spanning a particular exon-exon junction.

in gff3 format. The bam format is a format for aligned RNA-seq reads. To this end, the raw reads from each sample were uploaded to the bioinformatics data processing platform Galaxy [40] from the European Nucleotide Archive (ENA). For each sample, the raw reads in FASTQ format were then mapped to the reference genome GRCh38 [41] using STAR [42]. STAR produces the required bam and bai format files as output.

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 saves 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. See Figure 4.7 for an example output of VOILA. VOILA also allows filtering of LSV according to type and we used this to obtain the estimated PSI values of all exon skipping LSVs. The filtered LSV along with the constitutive exons obtained from MAJIQ Builder were then used for processing as described in Section 4.3.2.

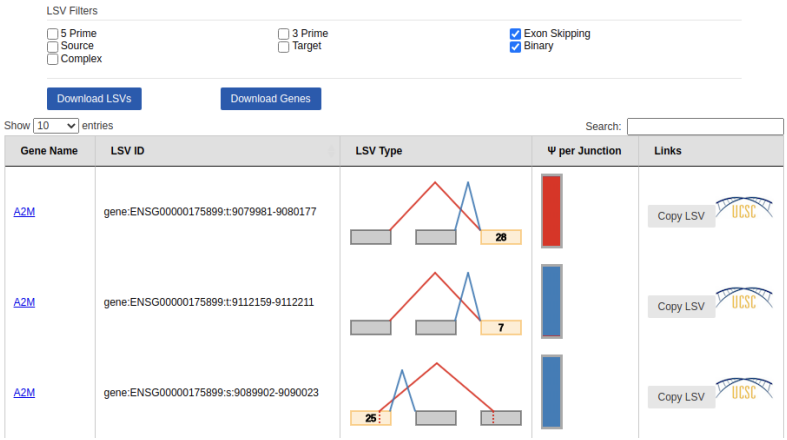

Figure 4.7: Example output of VOILA while filtering for LSV which only experience exon skipping and no other alternative splicing event.

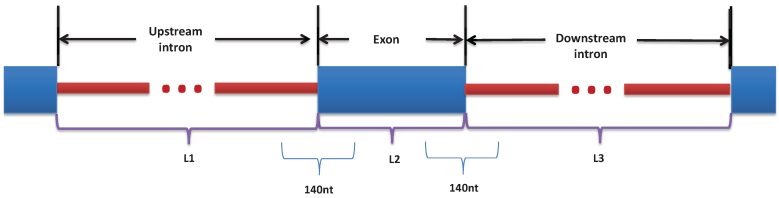

Figure 4.8: Schematic of model input for an exon-centric training sample [24]. The input are the 140 nucleotide region extracted around the exon start and exon end, as well as the normalized length of the exon and its flanking introns. The input for intron-centric training samples is analogue.

Our implementation of PSI estimation was applied to the GTEx dataset. SUPPA and MAJIQ were applied to the data from the HipSci repository.

4.3.2 Final sample processing

At this point, the datasets have been preprocessed such that we have the following information for each training sample that we want to create:

- 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*[1].

[1]Due to data limitations, we don't know the length of neighbouring exons when creating intron-centric training samples from GTEx data. In this case, the length of neighbouring exons is set to a constant 0.

Using this information, now either exon-centric or intron-centric training samples are created. Exon-centric training samples are created when the task of the network is to classify an exon as constitutive or not. Intron-centric training samples are created when the task of the network is to classify whether a junction is constitutive or not.

Using the chromosome information and the start and end coordinates, two 140 nucleotide window around the start and end coordinates were extracted from the human reference genome GRCh38 (see Figure 4.8). 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 nucleotide is converted to its reverse complement. This mirrors biology as the spliceosome also observes the exon start and end sites as switched and reverse complemented between the + and - strand. Presumably, this means the models 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 due to the fundamental functional differences between them and the autosomes (e.g. there only exists one copy of the sex chromosomes).

The extracted nucleotides were one-hot encoded as four dimensional vectors. Specifically, adenine 'A' is 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 encoded nucleotides.

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

As in previous work [24] [43], we include the lengths of flanking exons or intron and the length of the exon or intron itself as additional input features. For brevity, we often refer to these exon and intron lengths as the length features. The inclusion of the length features is motivated by the observation that exon and intron lengths are correlated with splicing behaviour [44] [45]. In particular, alternatively spliced exons

tend to be flanked by longer introns. This may be due to long intronic sequences making it more challenging for the spliceosome to locate the exon or it plainly being more likely that novel exons originate within long introns [46].

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 by the mean length and standard deviations of internal exon ($\mu = 145, \sigma = 198$) and introns ($\mu = 5340, \sigma = 17000$) in the human genome [47] [48].

In some of the datasets there are significantly more constitutive, than non-constitutive training samples. For instance, the HexEvent dataset contains roughly three times as many constitutive as alternatively spliced samples. In these cases, we rebalanced the datasets by including each alternatively spliced sample multiple times until the class imbalance was less than two-to-one.

The end results is that a single training sample for the model contains two 140x4 one-hot encoded matrixes and three normalized length values. This is the model input. The associated ground truth with each sample is the scalar 1 if the respective exon or junction is constitutive, and 0 if not.

4.3.3 Dataset statistics

Table 4.1 shows the number of obtained training samples in each datasets. Among the exon-centric datasets, the dataset based on HEXEvent contains the most samples, followed by the dataset based on HipSci data processed with MAJIQ, followed by the dataset based on GTEx data and, lastly, followed by the dataset based on HipSci and processing with MAJIQ. The HEXEvent-based and HipSci MAJIQ-based datasets contain comparatively more training samples because they also contain constitutive non-cassette exons, whereas the other datasets don't. Unlike between samples from neuron and iPSC cell lines, there are significant cross-tissue differences between the number of training samples in GTEx-based datasets: the

cerebellum tissue datasets contain roughly twice as many training samples as the heart tissue datasets. This is a result of the cerebellum tissue containing 10,000 highly-expressed whereas the heart sample contains only 5,000 highly expressed genes. Surprisingly, there are 17 million exon-exon junction reads from the heart sample compared to the cerebellum's 8 million exon-exon junction reads. The additional reads from the heart sample are concentrated in a few extremely highly expressed genes (e.g. gene ENSG00000198886 is expressed 1000-times as often as the median highly expressed gene) and while this concentration of reads also appear in the cerebellum tissue, it is less extreme.

The same cross-tissue observations also holds for the intron-centric datasets. The intron-centric datasets contains significantly more training samples than the exon-centric datasets because they are not constrained to cassette exons (which would lead to the expectation that the intron-centric datasets contain roughly twice as many samples as the exon-centric datasets), but contain all non-filtered junctions.

Name	Tissue	Type	Number of samples
HEXEvent	mixed	exon	50,918
GTEx	brain	exon	30,466
GTEx	cerebellum	exon	37,095
GTEx	heart	exon	19,257
GTEx	brain	intron	127,908
GTEx	cerebellum	intron	161,310
GTEx	heart	intron	81,659
HipSci SUPPA	neuron	exon	17,863
HipSci MAJIQ	neuron	exon	44,746
HipSci MAJIQ	iPSC	exon	48,489

Table 4.1: Type, tissue and number of training samples per dataset. Type refers to whether the dataset is intron- or exon-centric.

Figure 4.9 shows histograms for the exon-centric version of the four primary datasets versions. The histograms show that in each dataset the distribution of PSI values is bi-modal with the modes being around very rarely included ($< 5\%$) and very frequently included exons ($>95\%$). This aligns well with previous observations that alternatively spliced junctions and exons tend to be very frequently or very rarely

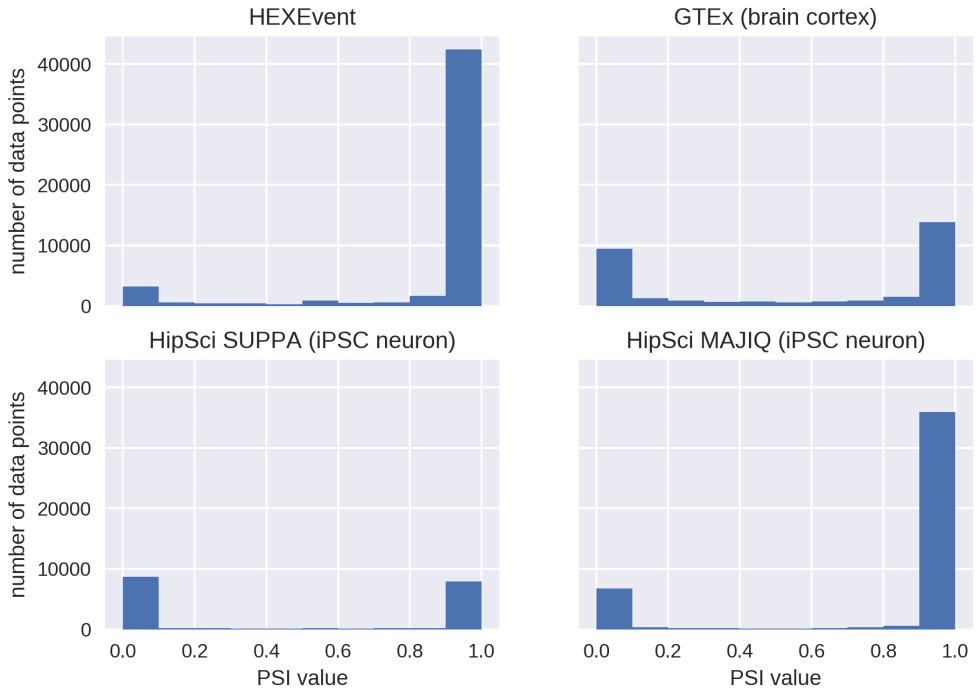

Figure 4.9: Histograms comparing the exon-centric versions of the four fundamentally different training datasets used. The shown GTEx dataset is based on samples from brain cortex tissue and the shown HipSci datasets are based on samples from iPSC cells differentiated to neurons.

included [23] [49] [50].

The lack of non-cassette constitutive exons is visible in theGTEx and HipSci SUPPA-based datasets containing significantly constitutive samples; the other two primary datasets contain roughly three times as many constitutive as non-constitutive samples.

In particular, HipSci SUPPA-based dataset contains an extremely low number of non-rarely or frequently included exons (only 1%) which is likely an artefact of its coarse, transcription-level estimation method.

4.4 Models

All three models are fundamentally split into two components:

1. The first component extracts the most relevant features of the two input

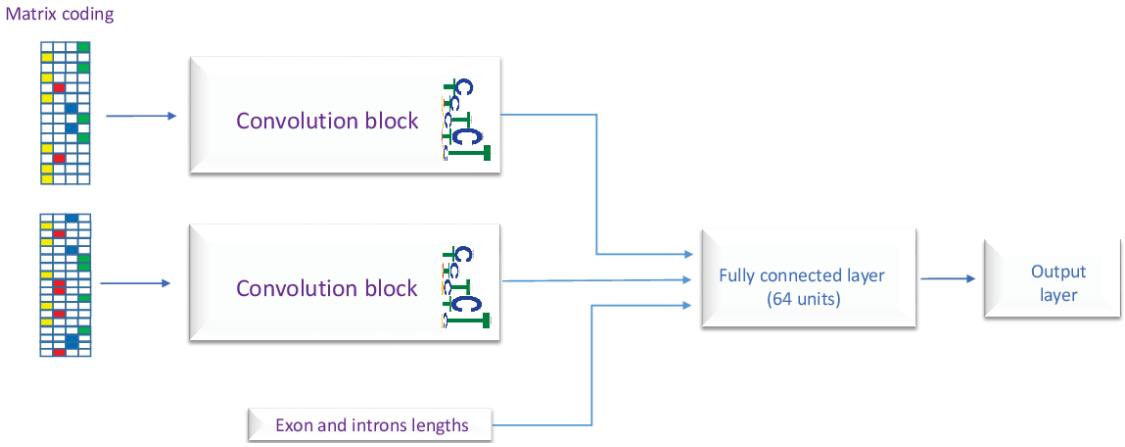

Figure 4.10: Architecture of DSC [24]. The one-hot encoded sequences are fed into two convolutional blocks which generate features that are concatenated with the exon and intron lengths. The sequence and length features are then used by a fully-connected layer to obtain the classification output.

sequences of nucleotides. Depending on the exact model, the feature extraction is based either on CNNs, word embeddings or BiLSTMs and an attention layer. The extracted features are concatenated along with the length features and fed as input to the second component.

2. 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].

4.4.1 DSC: CNN-based

This model is a reimplementation of the Deep Splicing Code (DSC) from [24] and the state-of-the-art model for alternative splicing classification.

DSC uses CNNs for feature extraction. 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 promising for the application to genomic data, that is, nucleotide sequences:

- Nucleotide sequences contain motifs (nucleotide sequence patterns conjectured to be biologically significant). Motifs can be represented as matrixes and indeed, they are frequently represented as position weight matrixes (PWMs) with particular weights. The kernels used in CNNs can be interpreted as PWMs with learnable weights, i.e., the model can automatically learn to recognize important motifs.
- Motifs are 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, DSC 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.

Concretely, each 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 [51], a ReLU activation layer [52] 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 [24]. 1D convolution layers are used.

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 [53] [54], applying Recurrent Neural Networkss (RNNs) to text as in Sequence-to-Sequence (seq2seq) learning [55] and leveraging attention as in the extremely powerful and deep Transformer models [56] [57].
In this work, we evaluate the application of Doc2Vec (a derivative of Word2Vec), Seq2Seq models and the attention mechanism to the task of alternative splicing classification. We don't evaluate the current state-of-the-art Transformer models. Albeit very potent, they usually require huge datasets in the order of millions of samples which aren't available for this task. Nonetheless, we do adapt the attention mechanism as known from Transformers for our task. In this way, we feel the evaluated models make use of the most important milestones using deep learning in NLP.

4.4.2 D2V: MLP-based

This model is a reimplementation of the model used for PSI regression on the mouse genome in [20]. The only architectural difference is that we possibly use a differently-sized MLP for classification, as no details on the MLP size and no publicly available implementation were given.

Introduction to word embeddings

Word2Vec is a neural network-based model which provides a continuously distributed (vector) representation for each word within a sentence [53][54]. The representation is chosen so that semantically and syntactically similar words have similar representations. The key idea through which Word2Vec achieves this is by representing words based on the context they appear in. To obtain the word representations, a shallow neural network is trained on a large corpus of unlabelled text with a loss which incentivices the model to learn such a representation.

Trained Word2Vec models have been shown to recover semantic and syntactic relationships. Keeping in mind that each word is represented as a numerical vector, the following equation usually holds on trained Word2Vec models: $\vec{King} - \vec{Man} + \vec{Woman} = \vec{Queen}$ where $\vec{x}$ represents the vector representation or embedding of word x.

Doc2Vec is an extension of Word2Vec which uses the same principles, but can handle variable length texts (such as paragraphs and complete documents) [58] [59]. 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. To this end, we obtained the complete human reference genome GRCh38 from UCSC [60].

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 [48], we split the genome into sequences of 28,000 nucleotide sequences. Tests with splitting the genome into sequences of length 10,000 that effect of this choice on model performance is negligible.

Using k-mers

Naively applying Word2Vec or Doc2Vec to genomic data would mean treating each nucleotide as a word. 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) is not unique enough and occurs in too many contexts to obtain meaningful representations. Therefore, it is desirable to embed multiple characters or multiple sequences, that is a word or a k-mer. While a sentence can easily be split into words, the split of a nucleotide sequence into k-mers is not as clear-cut: we don't know the functionality of most

nucleotides and therefore can't split a sequence into semantically meaningful units. Previous studies have explored this issue [61] [62], and found that the compromise of splitting each nucleotide sequence into overlapping 3-mers is a good solution.

Overlapping means that in the case of the sequence 'AACGAT' the resulting overlapping 3-mer sequence is 'AAC', 'ACG', 'CGA' and 'GAT'. Following this recommendation, we split the 28,000 nucleotides long pre-training sequences into 27,998 overlapping 3-mers.

Pre-training of word embeddings

The pre-training for Word2Vec and Doc2Vec uses a shallow neural network with one hidden layer. During a pre-training epoch, each word is once in the center of a sliding window or context window. One of two techniques is typically used for pre-training: either continuous bag-of-words (CBOW) or skip-gram. If CBOW is used, all words in a window except the current word are given as input and the target output is the current word. 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. A typical window size is 5 words. 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 4.11.

There is no clear choice for when which training method should be used for Word2Vec or Doc2Vec as they 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 a Doc2Vec model to output a 100-dimensional embedding for each document.

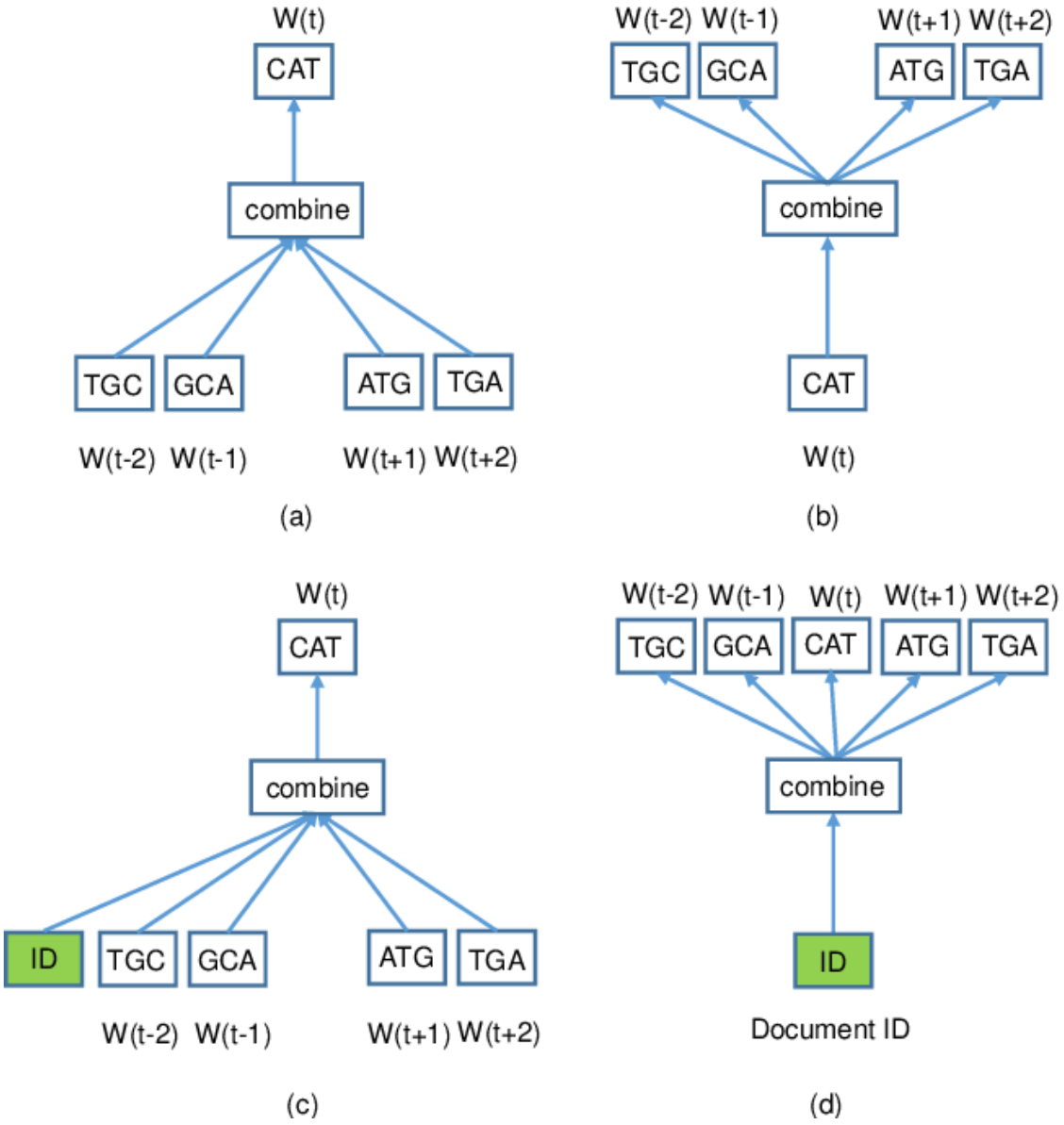

Figure 4.11: The four possible training algorithms for w2v and d2v models [20]: (a) CBOW, and (b) skip-gram for w2v, and (c) DM and (d) DBOW for d2v.

Obtaing the embeddings

After pre-training, the embedding for a given word or document is then the value of the hidden cells of the shallow MLP using during pre-training[2]. In the case of 100-dimensional embeddings, this means that the hidden layer contains 100 neurons.

[2]In practice, the words in a corpus are one-hot encoded and the value of the hidden layers doesn't need to be computed. Recalling that the weights of the hidden layer are represented as a $N \times d$ matrix, only the values in the d-dimensional column which belongs to the one-hot encoded word need to be retrieved. All other columns will be multipled by 0. N represents the size of the vocabulary (64 for us) and d the number of embedding dimensions (100 for us).

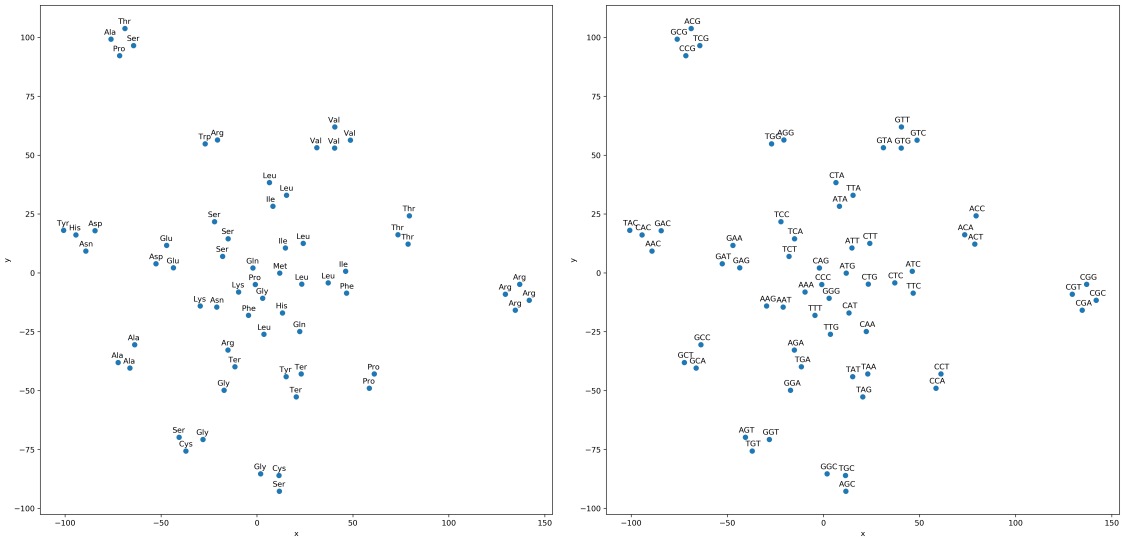

Figure 4.12: The 2-dimensional embeddings obtained by applying t-SNE to the 100-dimensional learned representations of all 64 possible 3-mers. Left shows the codons encoded by the 3-mers, right shows the actual 3-mers.

The intuition behind this is that the models learned a representation which is helpful for the prediction task the model was trained on. Since the prediction task was either to reconstruct a word or document given its context or vice-versa, the representation for a given word or document should encode its context. Thus, the main idea behind Word2Vec and Doc2Vec, that a word or document is defined by the context in which it occurs and its representation should encode this context, is achieved.

Analyzing the obtained embeddings

We visualize the embeddings for all 64 possible 3-mers using Stochastic Neighbor Embedding (t-SNE) [63] in Figure 4.12.

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.

Putting it all together

The resulting model uses the trained Doc2Vec model to obtain a 100-dimensional embedding of the window around the exon start and exon end. These two embeddings, along with the length features, are then given to a shallow MLP for classification. We refer to this model as the D2V model.

4.4.3 BiLSTM + Attn

Seq2Seq

Sequence-to-sequence learning (Seq2seq) [55] 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 [64] or GRUs [65]. Encoder and decoder are jointly trained to maximize the probability of correct output. This framework has been particularly successful in machine translation (MT): for instance, Google announced in 2016 that it started using them for their MT [66]. Figure 4.13 visualizes the working of an encoder-decoder model.

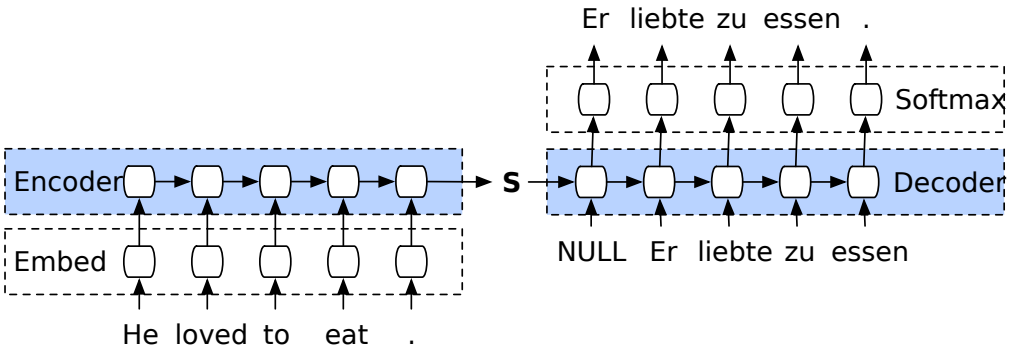

Figure 4.13: An encoder-decoder model [67] translates the English sentence 'He loved to eat' to the German sentence 'Er liebte zu essen'. The decoder takes the last hidden cell state of the encoder as initial input and generates the output token-by-token. The decoder takes its own previous output as prompt for the next output and stops generating output token when it generates a '.'-token.

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, [68] 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 [56] [57] [69] 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 [68] or multiplicative attention [56]) 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?

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 [70] have shown that otherwise, the model might suffer from limited generalization performance caused by the sparsity of the one-hot encoded representation.

4.4.4 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.

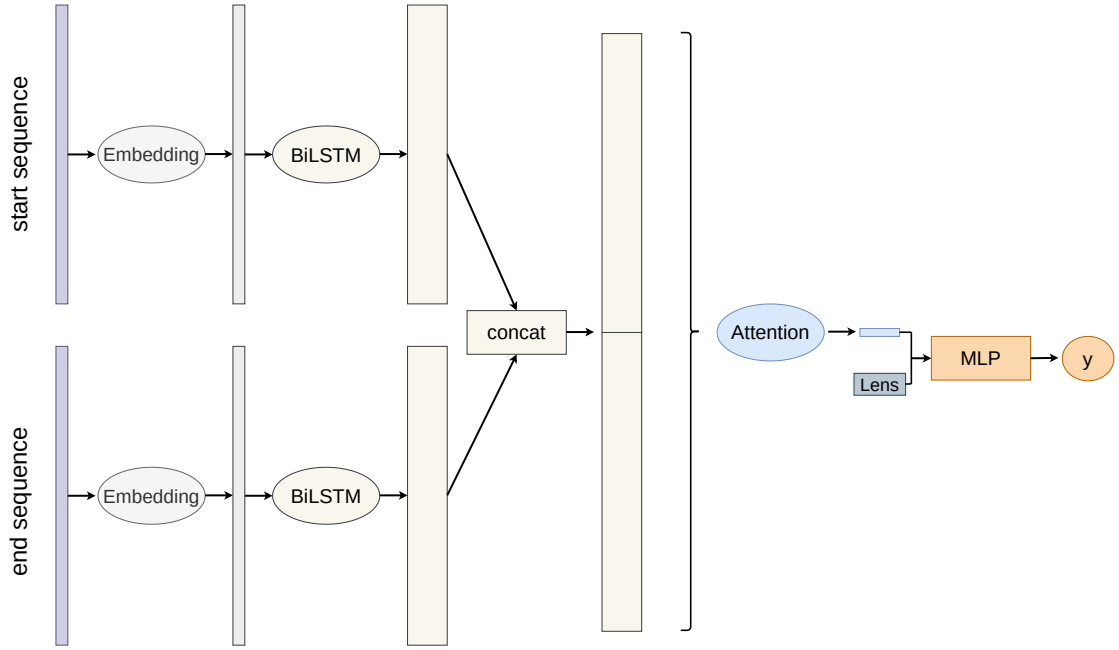

Figure 4.14: Architecture of the new attention-based model we introduce.

heads	no query	conv	μ	σ
0	0	0	0.832	0.025
0	0	1	0.860	0.021
0	1	0	0.851	0.013
0	1	1	0.818	0.019
1	0	0	**0.865**	0.011
1	0	1	0.824	**0.010**
1	1	0	0.852	0.017
1	1	1	0.846	0.017

Table 4.2: Results of evaluating the extensions to the Attention mechanism given as AUC. heads = multiple attention heads were used, no query = no conceptual query vector was used, conv = an additional convolution operation was applied to the key and value matrices. 0 codes for the extension not being used, 1 codes for the extension being used.

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 4.4.2, the information contained in a single nucleotide is very low. This motivated [71] 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 [71]. 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 [71]:

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 [71], 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 [72] 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.

4.5 Training and implementation details

4.5.1 Implementation

All models, except the Doc2Vec network, were implemented using the PyTorch library (version 1.5.0) [73]. It was chosen (over TensorFlow) as the main author already had more experience with PyTorch. The Doc2Vec model was implemented using the gensim library (version 3.8.3) [74]. 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 4.3.1 and 4.3.1. The latest versions of SUPPA (SUPPA v2.3) and MAJIQ (MAJIQ v2.0) available as of July 2020 were used. 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 datasets and 3 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]

4.5.2 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 75 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:

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

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 [24], 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.

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 4.3: Hyperparameters of the main models

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

The hyperparameters used for training the models are given in table 4.3. More details on the Doc2Vec pre-training procedure and hyperparameters values are given in the appendix 7.0.1. The DSC, D2V and BiLSTM + Attn models respectively contain 20,001, 6,801, and 66,861 trainable weights. In absolute terms, all of these model are still small compared to models known from Computer Vision and Natural Language Processing. The number of training weights is also reflected in the training times with the models respectively taking between 5 to 30 minutes, 1 to 5 minutes and 15 to 75 minutes to train once (depending on dataset).

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 [24] and introduced in 4.2.1. 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 [24] (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

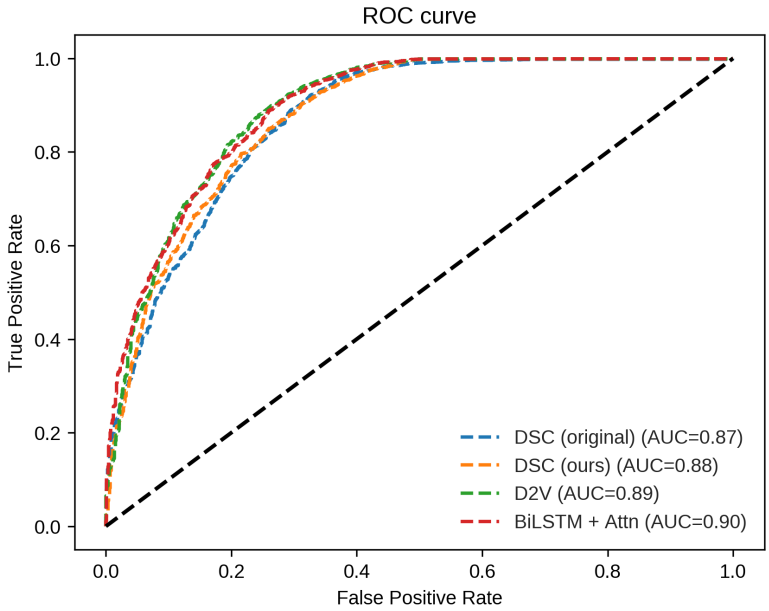

Figure 5.1: Comparison of the ROC curves of the three main models (as well as the original implementation) on the HEXEvent dataset. The values of the original implementation were obtained by rerunning the training on the publicly available implementation.

differences between TensorFlow / Keras (their implementation) and PyTorch (our implementation). Notably, when reevaluating the publically available original 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 improvement
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. The relative performance improvement (from adding the length features) was computed as $\frac{AUC_{lengths} - AUCno\_lengths}{AUC_{lengths} - 0.5}$. Computing it this way accounts for the baseline AUC of random guessing being 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 [24] 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 4.2.1, 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.

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 [24]. 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 [24]'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 [23], Random Forests [75] [43], Decision Trees [76] or AdaBoost-based algorithms [77]. Due to these data quality issue, we try to construct an alternative dataset.

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,

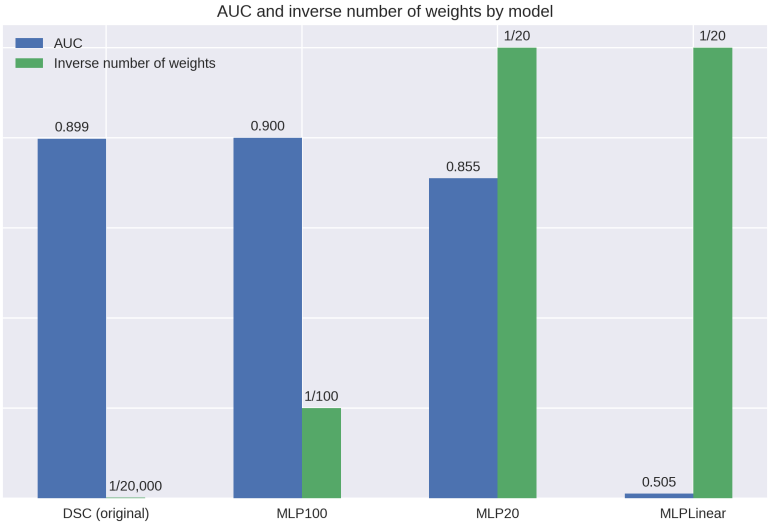

Figure 5.2: Stress testing the HEXEvent dataset used in [24]. The graph shows the performance as well as the inverse of the respective model sizes used.

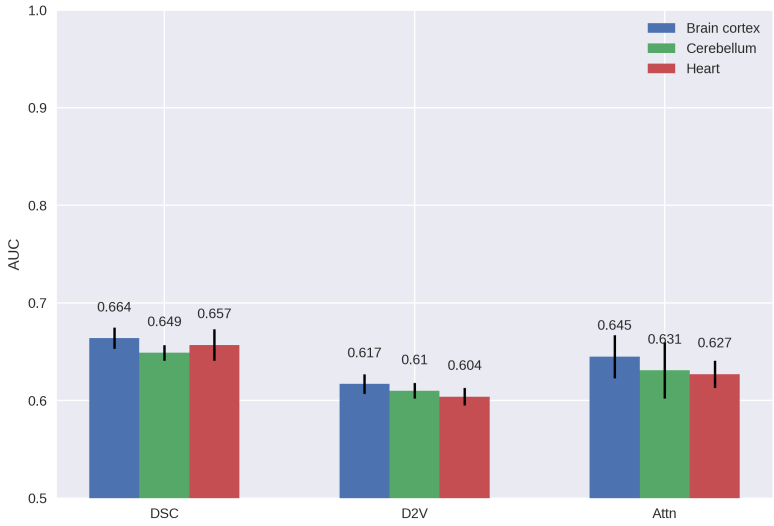

Figure 5.3: Performance on the GTEx-based exon datasets for different tissues. The error bars give the standard deviation across all cross-validation runs.

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.

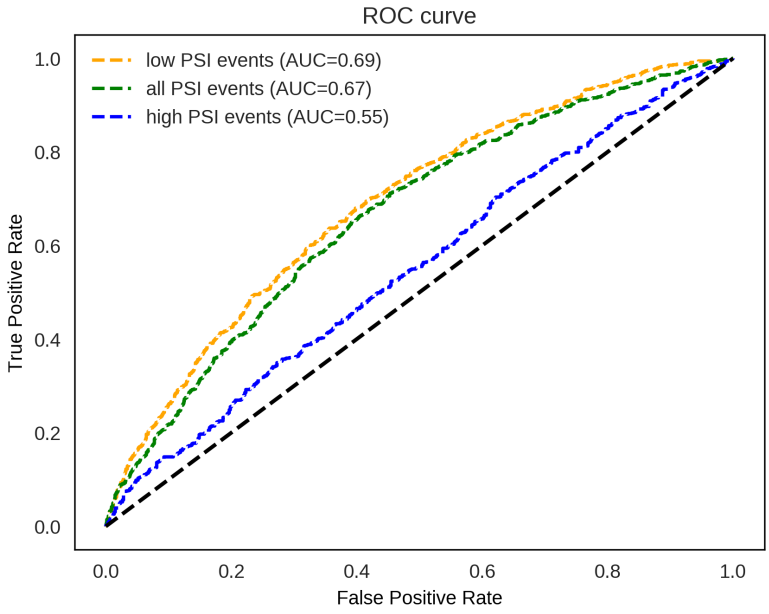

Figure 5.4: Bar chart of the performance on the GTEx-based exon datasets for different tissues. The error bars give the standard deviation across all cross-validation runs.

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 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 [24].

—- 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

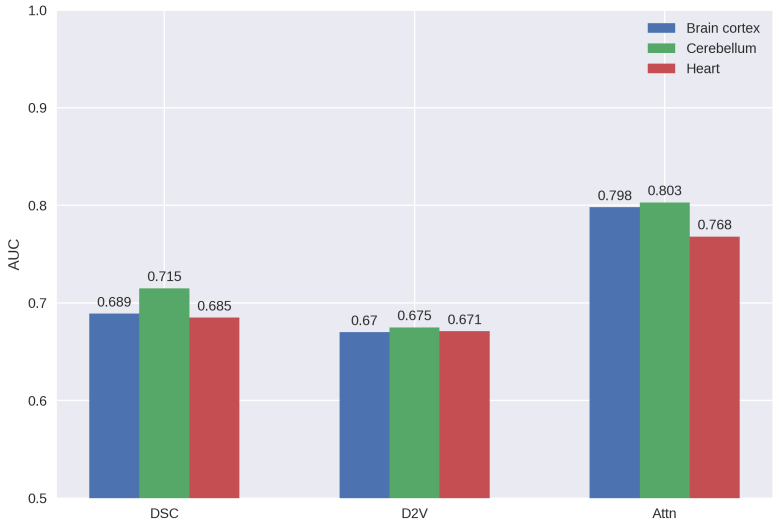

Figure 5.5: Performance on the GTEx-based junction datasets for different tissues.

TODO: motivation for using junction-centric datasets 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.

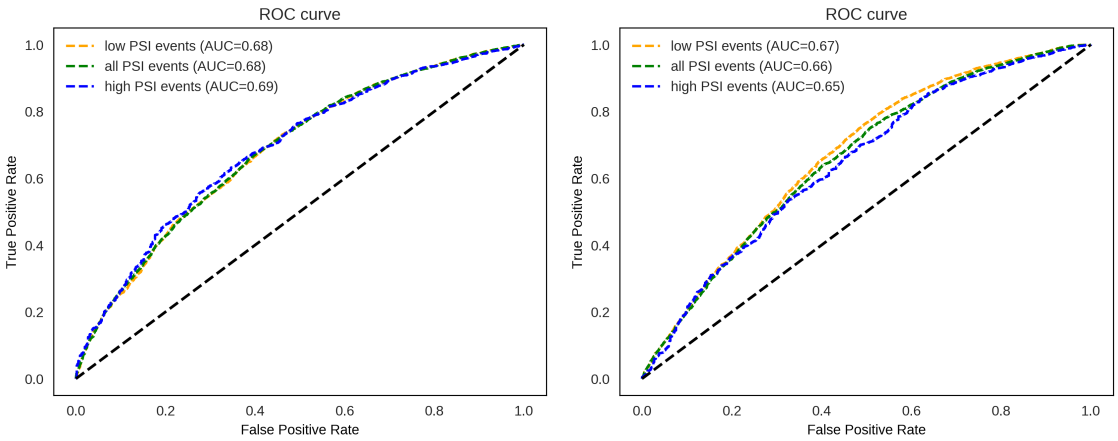

Figure 5.6: Left: ROC curve of the DSC model on heart tissue. Right: ROC curve of the Attn model on heart tissue.

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 4.3.1, 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 distoring the results in this section. 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

Figure 5.7 shows poor model performance across the board. The predictive power of our models on the SUPPA-based HipSci dataset is worse than on the GTEx dataset and roughly equal to performance the HEXEvent dataset without length features. This indicates that

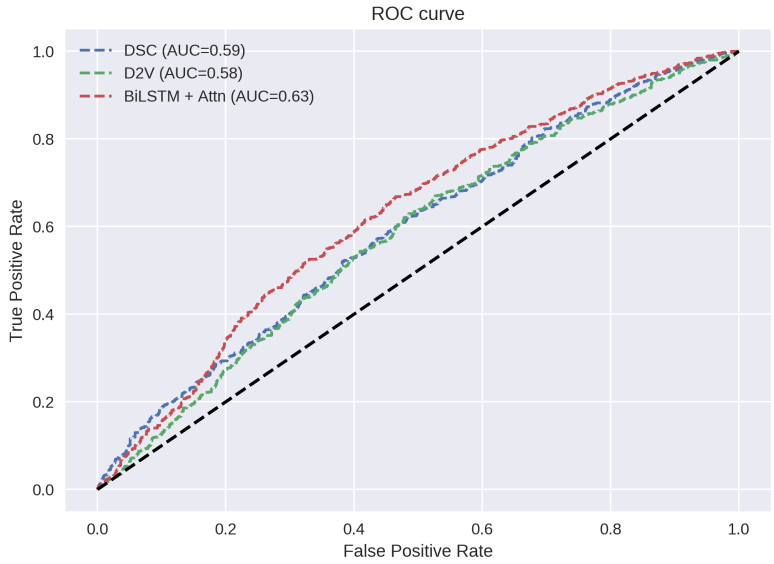

Figure 5.7: Comparison of the ROC curves of the three main models on the HipSci dataset derived by using SUPPA.

1. either we arrive at a strong dataset here and that constitutive exon classification is more challenging than so far anticipated,
2. or this dataset suffers from data quality issues which make constitutive exon classification very challenging for the models.

Among these two, 2. is more likely: in retrorespect, SUPPA was not the best choice for dataset processing. It operates at a transcript level and therefore implicitly relies on the assumption that all transcripts of a given gene are known - this is often not the case. Additionally, it does not make it possible to generate a list of non-cassette constitutive exons, therefore roughly halving the size of the training set and the number of samples the model can learn from. Its PSI estimation is very simplistic and doesn't account for multiple of the issues highlighted in 4.3.1. Thus, we now turn to data processing with MAJIQ in hopes of better results.

5.3.2 MAJIQ with iPSCs differentiated to neurons (exons)

All models perform better than in previous datasets, but in contrast to previous datasets we also observe stark disparities between the models (see Figure 5.8). While DSC outperforms D2V by a significant margin, it is outperformed by a

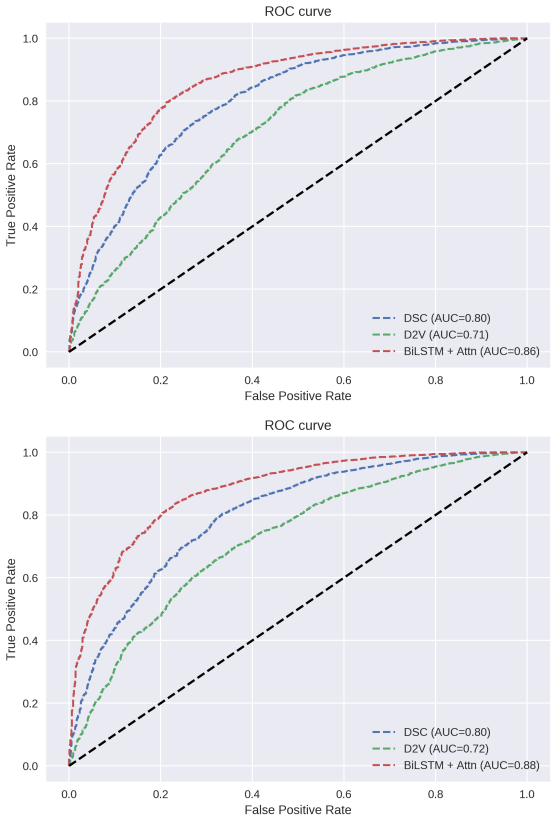

Figure 5.8: Left: ROC curves on dataset based on processing with MAJIQ and iPSC cells differentiated to neurons. Right: ROC curves on dataset based on processing with MAJIQ and undifferentiated iPSC cells.

similar margin by BiLSTM + Attn. The dataset based on undifferentiated iPSC samples seems to be slightly easier for the models, with AUC values across the board increasing by roughly 2%. At a closer look at the dataset distribution.... Applying the same test of evaluating the models with and without the length features, the results are very promising.

Model name	Only sequences	Sequences + lengths	Relative performance improvement
DSC	0.811	0.808	-0.010
D2V	0.665	0.715	0.233
BiLSTM + Attn	**0.853**	**0.865**	0.033

Table 5.2: Performance on the dataset based on processing with MAJIQ and iPSC cells differentiated to neurons with and without length features given. Note that the relative performance drop was computed with reference to the baseline AUC value of 0.5.

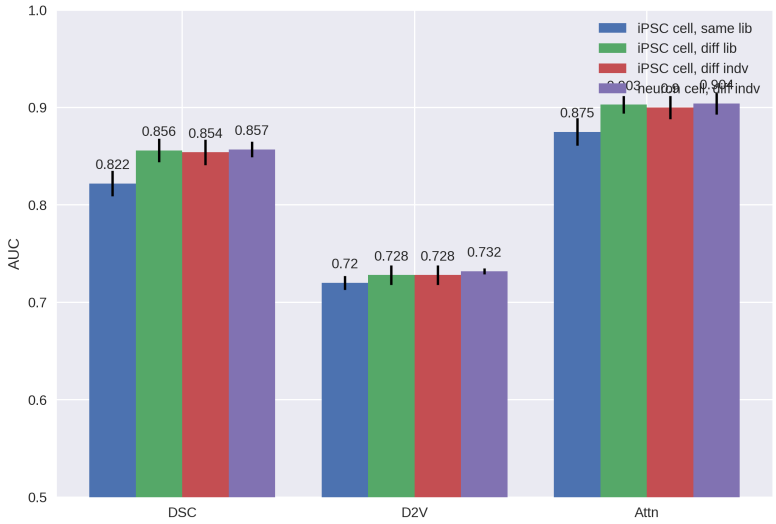

Figure 5.9: Performance when training models on one HipSci dataset and testing on the same dataset as well as three others. Within the bars of one model, going further right means using a dataset which is less similar to the dataset the model was trained on (the left-most bar display performance when testing the model on the dataset it was trained on). 'lib' refers to a library or biological sample from the same individual, 'indv' refers to the indivual the sample was taken from, 'diff' abbreviates different. 'iPSC cell' refers to a dataset based on RNA-seq data from undifferentiated iPSC cells while 'neuron cell' refers to iPSC cells differentiated to neuron cells.

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

- Extremely surprising results; it is better to train a dataset on undifferentiated tissue than training it on differentiated tissue –> def check data processing for this again lol, make sure that I actually shuffle data. this is too weird to be true

- constitutive exons are shared across all datasets - majiq builder builds up initial splice graph using all samples – some confounding there for sure - for this sort of analysis, prediting differential splicing itself would be more apt

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

continuous psi prediction too

interpretation of F1 score

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

6

Discussion / Conclusion

Cor animalium, fundamentum eſt vitæ, princeps omnium, Microcoſmi Sol, a quo omnis vegetatio dependet, vigor omnis & robur emanat.

The heart of animals is the foundation of their life, the sovereign of everything within them, the sun of their microcosm, that upon which all growth depends, from which all power proceeds.

— William Harvey

7 Appendix

Appendices are just like chapters. Their sections and subsections get numbered and included in the table of contents; figures and equations and tables added up, etc.

Accession Numbers of HipSci data

The ENA Accession Numbers of the 25 biological replicates belonging to sensory neuron cell lines [32] are ERR177-: 5544, 5551, 5552, 5554, 5594. 5595, 5596, 5598, 5600, 5601, 5631, 5634, 5637, 5638, 5640, 5641, 5643, 5644, 5684, 5685, 5686, 5687, 5688, 5689, 5693. While all of these were used in MAJIQ Builder process (and thus contributed to the constitutive exons), only the sample with Accession Number ERR1775544 was used with the MAJIQ PSI step (and thus determined the alternatively spliced exons).

The ENA Accession Numbers of the 20 biological replicates belonging to undifferentiated iPSC cell lines [28] are ERR-: 914342, 946968, 946976, 946983, 946984, 946990, 946992, 946994, 947011, 1203463, 1243454, 1274914, 1274917, 1724696, 1724699, 1743789, 2039345, 2039336, 2278244 2278245. Similarly, all of the above biological replicates were used in the MAJIQ Builder process. Only the samples with Accession Numbers ERR946992, ERR946984 (same cell type and donor as

ERR946984, but different cell line), and ERR946968 (same cell type, but different donor) were used with MAJIQ PSI.

7.0.1 Additional Doc2Vec training details

Training method	DM
Embedding dimensions	100
Corpus	Human Genome GRCh38
Window size	5
Minimum count	5
Negative sampling	5
Epochs	5

Table 7.1: Exact hyperparameters used for training Doc2Vec model.

The hyperparameters used during pre-training are given in table 7.1. Except for the number of epochs, these are the same as in the baseline paper [20]. We reduced the number of epochs from 20 to 5, as initial tests showed no performance difference between these two values. However, while [20] don't mention what Doc2Vec implementation they use, almost all of these parameters are the same as the default parameters from the gensim library. Therefore, we believe that these parameters weren't fine-tuned very intensively.

Two hyperparameters, not yet introduced, are mentioned in Table 7.1. Although these hyperparameters aren't very impactful when training on genomic data, we mention them for completeness (as they are also mentioned in [20]):

- The minimum count parameters eliminates all words which occur fewer than the minumum amount from the corpus. Infrequent words don't have enough examples to allow the model to learn a good representation of them. Additionally, while the individual words might uncommon, there might be a lot of them, leading to an additional computational effort.
- Negative sampling [54] is a technique to reduce the computational cost of backpropagation steps. The size of the weights in the Word2Vec or Doc2Vec can be easily reach millions of learnable weights with a medium

sized vocabulary: for 10,000 words in the vocabulary and 300-diimensional embedding, the matrix representing the hidden weights already has 3 million weights. When negative sampling is enabled, by default only the weights connected to the word we want to predict will be updated via backpropagation. Additionally, a certain number of negative samples, words which the network shouldn't predict, are randomly chosen and their weights updated too. This dramatically reduces the computational effort for each backpropagation step, since only the weights of very few words in the vocabulary are updated.

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

— Sir Isaiah Berlin

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.

[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] Qiang Xu and Christopher Lee. "Discovery of novel splice forms and functional analysis of cancer-specific alternative splicing in human expressed sequences". In: *Nucleic acids research* 31 (Oct. 2003), pp. 5635–43.

[11] 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.

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

[13] 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.

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

[15] 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.

[16] 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.

[17] 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.

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

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

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

[21] 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.

[22] 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.

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

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

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] Shivashankar H. Nagaraj, Robin B. Gasser, and Shoba Ranganathan. “A hitchhiker’s guide to expressed sequence tag (EST) analysis”. In: *Briefings in Bioinformatics* 8.1 (May 2006), pp. 6–21.

[30] M Bonaldo, Greg Lennon, and Marcelo Soares. “Normalization and Subtraction: Two Approaches to Facilitate Gene Discovery”. In: *Genome research* 6 (Oct. 1996), pp. 791–806.

[31] Y tambe. *File:Induction of iPS cells.svg*. [Online; accessed 23-August-2020]. 2007. URL: `https://commons.wikimedia.org/wiki/File:Pre-mRNA_to_mRNA_MH.svg`.

[32] Jeremy Schwartzentruber et al. “Molecular and functional variation in iPSC-derived sensory neurons”. In: *bioRxiv* (2017). eprint: `https://www.biorxiv.org/content/early/2017/01/06/095943.full.pdf`.

[33] Yuval Benjamini and Terence P. Speed. "Summarizing and correcting the GC content bias in high-throughput sequencing". In: *Nucleic Acids Research* 40.10 (Feb. 2012), e72–e72. eprint: https://academic.oup.com/nar/article-pdf/40/10/e72/25335311/gks001.pdf. URL: https://doi.org/10.1093/nar/gks001.

[34] Wei Zheng, Lisa Chung, and Hongyu Zhao. "Bias detection and correction in RNA-Sequencing data". In: *BMC bioinformatics* 12 (July 2011), p. 290.

[35] GTEx Consortium. *The GTEx Project: Documentation.* [Online; accessed 26-August-2020]. 2020. URL: https://gtexportal.org/home/documentationPage.

[36] 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.

[37] Rob Patro et al. "Salmon provides fast and bias-aware quantification of transcript expression". In: *Nature methods* 14.4 (2017), pp. 417–419.

[38] Gael P Alamancos et al. "Leveraging transcript quantification for fast computation of alternative splicing profiles". In: *Rna* 21.9 (2015), pp. 1521–1531.

[39] 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.

[40] 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.

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

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

[43] Ying Cui, Meng Cai, and H. Stanley. "Comparative Analysis and Classification of Cassette Exons and Constitutive Exons". In: *BioMed Research International* 2017 (Dec. 2017), pp. 1–8.

[44] Seung Gu Park and Sridhar Hannenhalli. "First intron length in mammals is associated with 5' exon skipping rate". In: *bioRxiv* (2015).

[45] Kristi L. Fox-Walsh et al. "The architecture of pre-mRNAs affects mechanisms of splice-site pairing". In: *Proceedings of the National Academy of Sciences* 102.45 (2005), pp. 16176–16181.

[46] Sahar Gelfman et al. "Changes in exon–intron structure during vertebrate evolution affect the splicing pattern of exons". In: *Genome research* 22 (Jan. 2012), pp. 35–50.

[47] 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.

[48] 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.

[49] Jie Wu et al. "Splice Trap: A method to quantify alternative splicing under single cellular conditions". In: *Bioinformatics (Oxford, England)* 27 (Sept. 2011), pp. 3010–6.

[50] Shihao Shen et al. "MATS: a Bayesian framework for flexible detection of differential alternative splicing from RNA-Seq data". In: *Nucleic Acids Research* 40.8 (Jan. 2012), e61–e61. eprint: `https://academic.oup.com/nar/article-pdf/40/8/e61/25334363/gkr1291.pdf`. URL: `https://doi.org/10.1093/nar/gkr1291`.

[51] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[52] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines". In: *ICML*. 2010.

[53] 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]`.

[54] 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]`.

[55] 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]`.

[56] 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]`.

[57] 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]`.

[58] 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]`.

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

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

[61] Patrick Ng. "dna2vec: Consistent vector representations of variable-length k-mers". In: (Jan. 2017).

[62] Ehsaneddin Asgari and Mohammad RK Mofrad. "Continuous distributed representation of biological sequences for deep proteomics and genomics". In: *PloS one* 10.11 (2015), e0141287.

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

[64] 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`.

[65] 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].

[66] 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].

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

[68] 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].

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

[70] 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].

[71] 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.

[72] 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.

[73] 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.

[74] 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.

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

[76] Juan A. Botia et al. “G2P: Using machine learning to understand and predict genes causing rare neurological disorders”. In: *bioRxiv* (2018).

[77] 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.