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

1044935

Department of Computer Science

University of Oxford

Submitted in partial completion of the

MSc in Computer Science

Trinity 2020

Abstract

Alternative Splicing (AS) is a fundamental part of gene expression and its misregulation has been associated with up to 20% of all diseases, yet the regulatory mechanisms influencing it are still poorly understood. Better understanding of AS is in part driven by computational models which predict AS behaviour based on sequence information.

Here, we show that a dataset which was previously widely used for the quantification of AS is confounded such that 100 parameters models can match the performance of 20,000 parameter models, calling the meaningfulness of results using this dataset into question. To this end, we construct three new datasets, each based on a different processing method, and show that only one of them provides high enough data quality for our task. We develop a new Deep Learning model which is the first to introduce the Attention mechanism to AS prediction. Evaluating our newly proposed model, we reimplement two models from the literature and find that it outperforms the previous state-of-the-art model by 15%. Finally, we interpret our model and show that it attends to biologically plausible motifs.

Thus, our work shows that previous results have been based on critically confounded data, provides better datasets upon which future work can build and introduces a new state-of-the-art model.

Contents

Glossary **v**

1 Introduction **1**

2 Background **4**

3 Related work **9**

4 Methods **13**

4.1 Task formulation 13
4.2 Datasets 14
4.2.1 HEXEvent database 14
4.2.2 GTEx 15
4.2.3 HipSci 16
4.3 Data processing 17
4.3.1 Estimating PSI 17
4.3.2 Final sample processing 27
4.3.3 Dataset statistics 30
4.4 Models 32
4.4.1 DSC: CNN-based 32
4.4.2 D2V: MLP-based 35
4.4.3 RASC: BiLSTM and Attention-based 40
4.4.4 Further modifications of the attention mechanism 45
4.5 Implementation, training and evaluation 50
4.5.1 Implementation 50
4.5.2 Training 51
4.5.3 Evaluation 52

5 Results **54**

5.1 HEXEvent dataset 54
5.2 GTEx-based datasets 58
5.2.1 Exon-centric datasets 58
5.2.2 Intron-centric datasets 60

5.3 HipSci SUPPA datasets 62
5.3.1 Dataset based on sensory neuron cell lines 62
5.4 HipSci MAJIQ datasets 64
5.4.1 Dataset based on sensory neuron and iPSC cell lines 64
5.4.2 Evaluation of the additional attention modifications 66
5.4.3 Sensitivity and specificity analysis 68
5.4.4 Cross-condition performance 69
5.4.5 Interpreting RASC 71
5.4.6 PSI Regression 73

6 Discussion / Conclusion **76**

7 Appendix **78**
7.1 Accession Numbers of HipSci data 78
7.2 Additional Doc2Vec training details 79

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.
Junction	Site in a mature mRNA transcript where two exons border each other.
Cis-regulatory element	Non-coding DNA region on the same DNA molecule as the gene ('close to') they regulate.
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	The Genotype-Tissue Expression project, a large repository of human-sequencing data which we use.
iPSC	induced pluripotent stem cells, mature cells which have been reprogrammed to an immature pluripotent (undetermined) state.
HipSci	Human Induced Pluripotent Stem Cell Initiative, a repository of iPSC-based data which we use.
Splicing Code	A (computational) model, describing the splicing process.
DSC	Deep Splicing Code, a model used for constitutive exon classification.

Why genes in pieces?
— Walter Gilbert [**whygenesinpieces**], in response
to the discovery of splicing

1 Introduction

"One gene, one protein" was the prevailing dogma for over 30 years. Yet this belief was turned upside down by the stunning discovery in 1977 that genes contain long non-coding sequences [**discoveryofsplicing**] [figure]. Instead, these non-coding sequences must be removed and only the coding sequences are joined together to form the blueprint of a gene product. This behaviour is not uniform and different sequences may be spliced together in different ways under different circumstances, allowing one gene to encode multiple proteins. Splicing and Alternative Splicing were discovered.

But "Why genes in pieces?". This question was posed by Walter Gilbert one year later [**whygenesinpieces**]. Gilbert answered his own question and proposed that splicing in essence provides a multiplier which speeds up the rate of evolutionary adaption: a single nucleotide mutation influencing splicing may now lead to the inclusion or exclusion of whole genomic regions and thus the generation of a novel gene product. Was Gilbert correct? 42 years later we are still uncertain, but some support for his ideas have been found [**whyrevisited**].

What we have gained in the past four decades of research, is a greater appreciation of the importance and the complexity of splicing. Misregulated alternative splicing has been shown to be a biomarkers for several forms of cancer [**cancer**] [**splicingcausescancer**] and about 50% of known disease-causing mutations have been shown to affect splicing [**50diseasessplicing**]. Yet, our understanding of splicing is still limited: the core splicing signals which we have found are estimated to only account for approximately 50% of splicing behaviour even in limited circumstances. The remaining behaviour is likely accounted for by the complex interactions of numerous cis-regulatory elements.

An important long-term goal is the development of a splicing code which describes the regulatory mechanisms governing splicing [**longtermcall**] and unifies them into a predictive model. Due to splicing's complexity, attempts at these splicing codes have been increasingly computational [**barash2010a**] and although many have been proposed, they still fail to explain a lot of the splicing behaviour we observe.

A parallel development has been the rise of Deep Learning enabled by the increasingly large available datasets and computing power [**deeplearning**]. Originating from Computer Vision , Deep Learning has brought about breakthroughs in tasks as disparate as the early detection of Alzheimer's disease [**alzheimerdeeplearning**] or estimating the demographic makeup of the US [**demographic**]. With the advent of next-generation sequencing, the amount of genomic data available has increased by a multi-fold making it a prime application domain for Deep Learning.

Our work lies at the intersection between the desire for better splicing codes and the revolution of many fields by Deep Learning. We develop a Recurrent Attentive Splicing Code (RASC) inspired by Deep Learning techniques known from Natural Language Processing (NLP). During evaluation, we find that we can replicate the performance of a 20,000 parameter splicing model using a 100 parameter model which takes no sequence information as input. Motivated by this finding, we set out to develop better datasets and construct three classes of datasets based on three different processing methods. We show that only one of them provides appropriate data quality and quantity for the training of Deep Learning-based splicing codes. Comparing RASC against two reimplemented splicing codes, we show that RASC outperforms the previous state-of-the-art splicing code for alternative splicing exon classification by 15%.

In summary, we make the following contributions:

- we show that a dataset used in previous research is critically confounded.
- we construct a new dataset to train Machine Learning-based splicing codes.
- we develop a new state-of-the-art splicing code.

The rest of the text is organized as follows:

- Chapter 2 succinctly introduces the required biological background required to follow the rest of this work. It also gives more detail about the importance of splicing.
- Chapter 3 reviews the splicing codes which have already been developed.
- Chapter 4 describes the dataset construction and the evaluated models in detail. It is by far the longest and most theoretical chapter due to us developing datasets and due to us modifying the Attention mechanism known from Transformer models.
- Chapter 5 presents the results of evaluating the models on a total of 16 different datasets. This includes the dataset whom we show to be critically confounded and the datasets for whose future us we advocate.
- Chapter 6 summarizes our findings and draws more general conclusions. We also give suggestions for future work.

2
Background

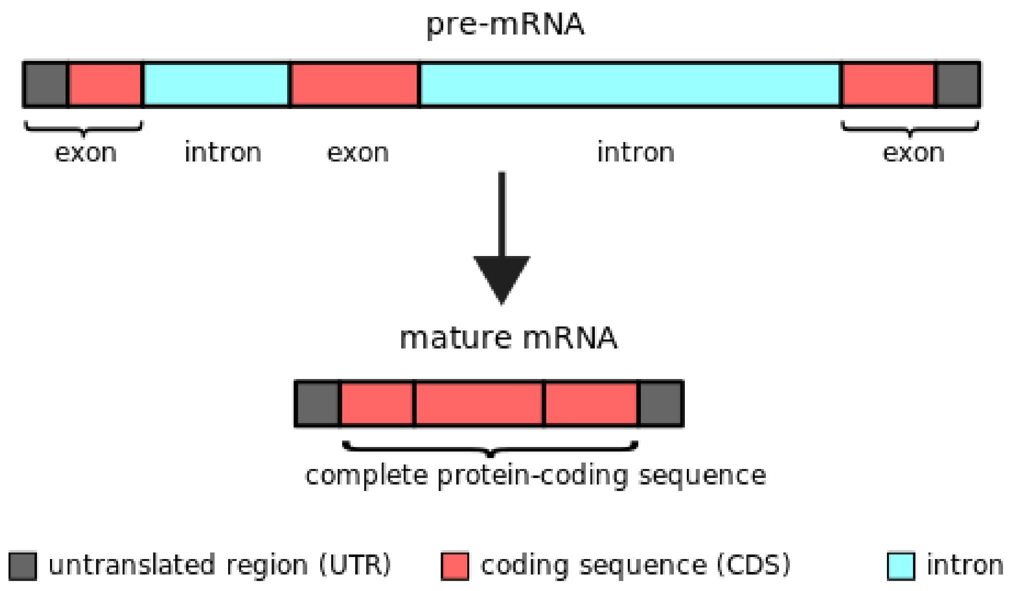


Figure 2.1: The process of splicing [**img:mrna**]. Introns are removed from the pre-mRNA to obtain the mature mRNA only consisting of exons. Apart from coding regions, exons may also consist of non-coding untranslated regions (UTRs). Like introns, UTRs influence gene expression.

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 (see Figure 2.1). The spliceosome is a complex molecular machine consisting of as many as 150 proteins [**splicing_current_perspectives**]. The previous location of an intron in the transcript where two exons adjoin each other is called a splice junction or splice site.

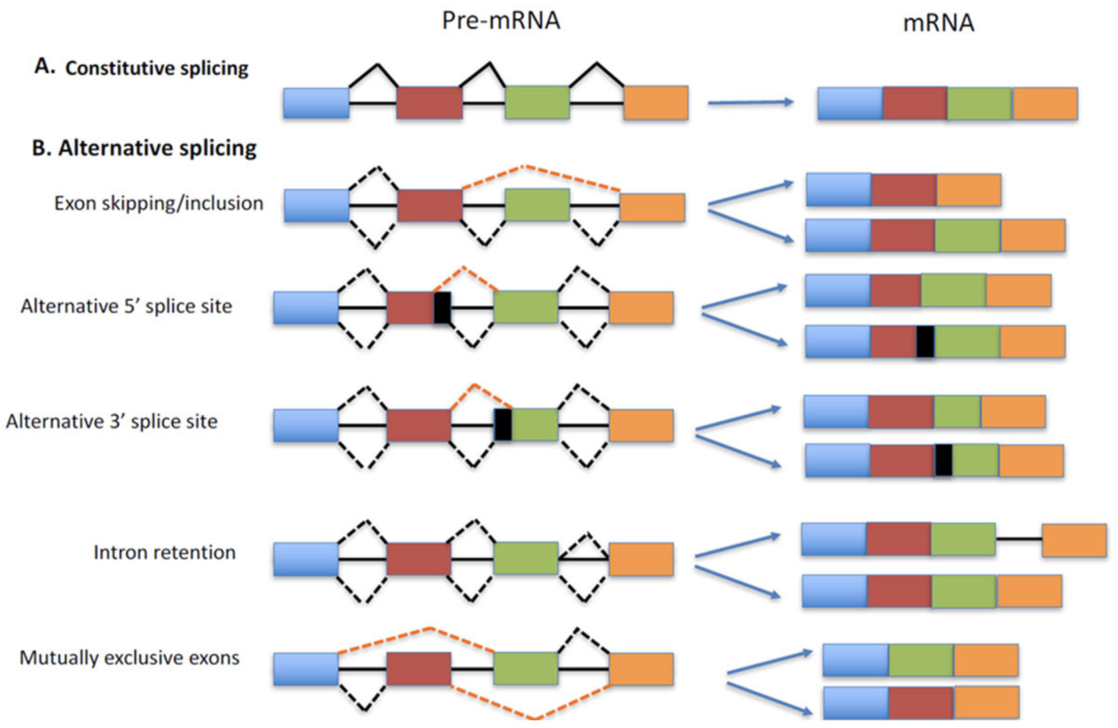


Figure 2.2: The most common forms of alternative splicing and the resulting different possible mature mRNAs [**img:altsplicingforms**].

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. Alternative splicing results in the generation of multiple transcripts from a gene. The most common types of alternative splicing in higher eukaryotes are [**commonsplicing1**][**commonsplicing2**]:

- Cassete exons are exons which 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 [**splicing_current_perspectives**].
- 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 visualized as falling to the right of the sequence). The 5' splice site or splice junction is the end of the exon towards the 5' end of the RNA strand (typically visualized as falling to the left of the sequence). 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 [**splicing_current_perspectives**].
- intron retention, that is, when an intron between exons is not spliced out. It accounts for roughly 5% of AS activity in higher eukaryotes [**splicing_current_perspectives**].

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 less common. Alternative splicing occurs in nearly all organisms that carry out pre-mRNA splicing as such as plants or animals and its frequency varies across organisms [**splicing_current_perspectives**].

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 [**splicing_current_perspectives**].

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. [**bretschneiderphdthesis**]

How is alternative splicing regulated?

Alternative splicing was discovered 40 years ago [**discoveryofsplicing**], 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 [**splicing_current_perspectives**]. 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 [**comparison**] Splice variants have also been shown to be biomarkers for multiple types of cancers [**cancer**] [**splicingcausescancer**]. As a result, alternative splicing has also been suggested as a biomarker and potential target for drug discovery [**drugdiscoverysplicing**].

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. [**bretschneiderphdthesis**] 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 [**barash2010a**][**barash2010b**]. 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.

[**barash2010a**] 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 (sometimes denoted Ψ) to describe splicing behaviour: PSI is defined as the proportion of transcripts out of all transcripts that contain a given exon [**psi**]. 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 ΔPSI. They were able to find novel regulators of key genes associated with diseases and to predict how genetic variants will affect splicing [**splicingcodegood1**] [**splicingcodegood2**]. 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. [**bnnsplicing**] 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 were 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 [**leung2014**]. Using a Deep Neural Network (DNN) with an autoencoder, they were able to improve upon the results achieved by BNN models. Albeit [**leung2014**] initially used a different dataset and a different task formulation than [**bnnsplicing**], [**jha**] were able to show that these improvements also lasted when directly comparing the models on the same dataset using the same task formulation. Furthermore, [**jha**] 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 [**leung2014**]. Adding these additional 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, [**d2vsplicing**] 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 [**inception**], 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. 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 [**splicingcodegood1**]. For predicting whether an exon is constitutive or not, features around the cassette exon itself have been shown to be the most critical [**featurearoundexonjunc**].

[**buschhertel**] used 262 features extracted from an exon and its 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. [**dsc**] 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.

4.1 Task formulation

We consider a classification and regression task. For the classification task the models classify an exon or junction as constitutive or not. For the regression task the models regress the PSI value of an exon or junction. The training samples comprise of exon or junction features helpful for classification or regression and the corresponding PSI value.

For the classification task, we classify all exons and junctions based on their PSI value before training on them. 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 $\geq 99\%$ as constitutive. For the regression we use the PSI values as training labels. For either task, the output of our models is a prediction $\hat{y} \in [0, 1]$.

We focus on the easier classification task at first and only evaluate the models on the more challenging regression task once we have achieved a good predictive power for the classification task.

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) [**dsc**] [**buschhertel**]. 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 [**hexevent**], data from the Genotype-Tissue Expression (GTEx) project [**gtex**] and data from the Human Induced Pluripotent Stem Cell Initiative (HipSci) [**hipsci**]. 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. ESTs 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 [**hitchhiker**]. Additionally, ESTs underpresent less frequent transcripts and as a result often only capture 50 - 65% of an organism's genes [**estunderrepresenttransripts**]. 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.

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

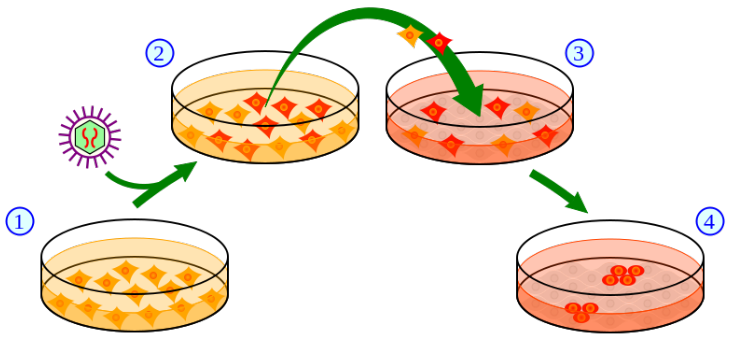


Figure 4.1: The four high-level steps taken to obtain iPSC cells [**img:ipscprocess**]: (1) Grow a cell culture of donor cells, (2) transduce the genes associated with reprogramming into the cells via viral vectors, (3) isolate the cells expressing the transduced genes (in red) and culture them according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

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 [**ipscneurons**]. 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 7.1.

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 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.

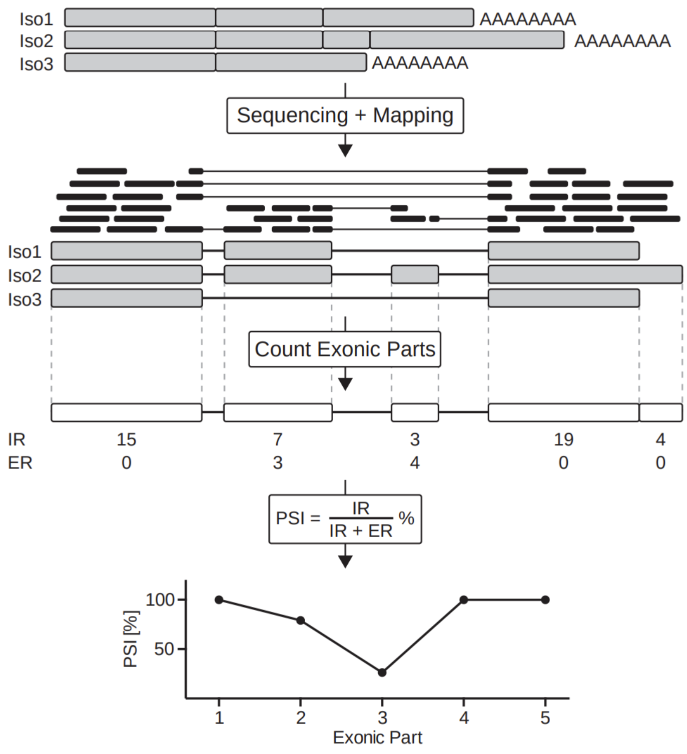


Figure 4.2: Process of estimating PSI based off RNA-seq reads [**berlinpsi**]. First, the reads are mapped to the transcript. Based on annotation files, the reads overlapping exon/intron junctions are classfied as including or excluding reads for a particular exon. Based on the observed IR and ER, the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

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 overall gene expression: 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 abundance 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 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

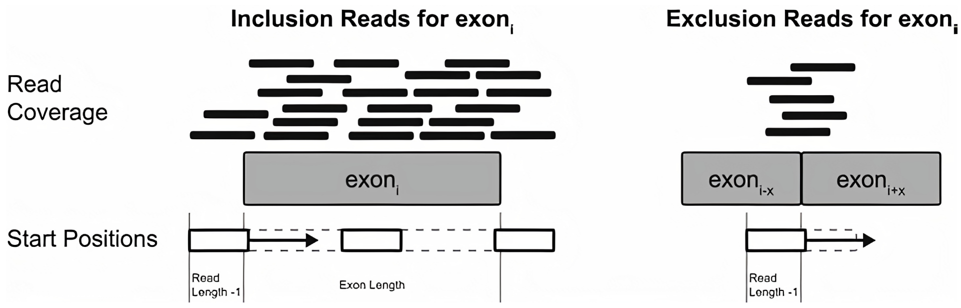


Figure 4.3: This graphic showcases the need for read length normalization when estimating PSI [**berlinpsi**]. An IR (left) can be located anywhere over the exon and around the junction. An ER (right) can only be located where it crosses the exon-exon junction. Thus, any PSI estimation which does not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

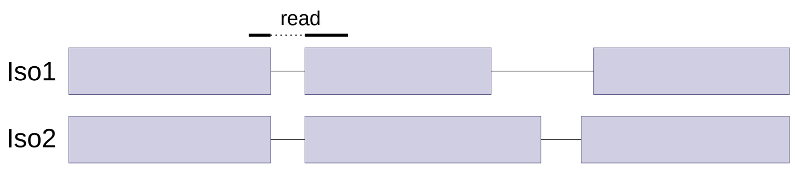


Figure 4.4: The read across the exon-exon junction may be attributed to isoform 1 or isoform 2. Consequently, it may also be attributed to the first or second splicing variant of the middle exon.

normalizing for the possible number of start positions of each read population [**berlinpsi**]:

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

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

(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.

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 [**gccontentbias**], gene length and dinucleotide frequencies [**rnaseqbiascorrection**]. 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 [**dsc**] 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 [**dsc**]. The EST data available from 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 the genome and are usually an artifact of sequencing errors [**dsc**] [**shortintrons**].

- 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 [**gtex_read_length**] 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. MAJIQ has also already been used for the construction of similar datasets before [**jha**].

SUPPA

SUPPA [**suppa2**] estimates the PSI value for each alternative splicing event based on transcript abundance. 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 [**salmon**] was used to quantify 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's main competitive advantage for PSI estimation lies in its speed and low memory overhead compared to other methods [**suppa**] (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

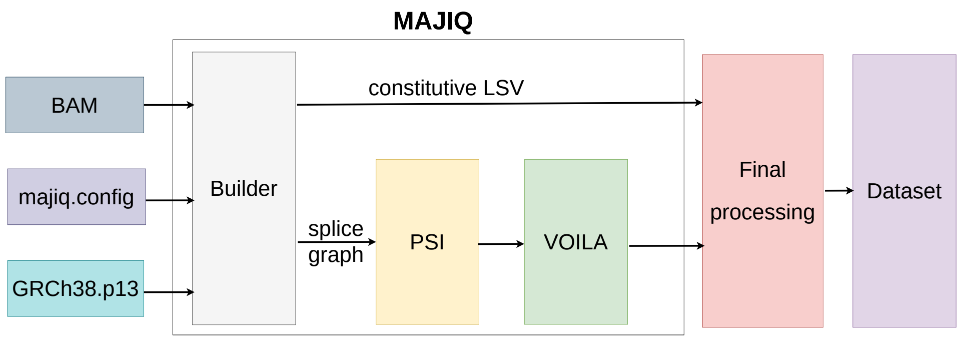


Figure 4.5: Data flow when using MAJIQ to create a training dataset.

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 [**majiq2**] 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.

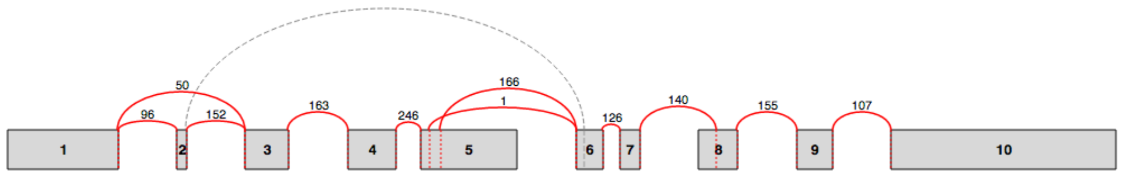
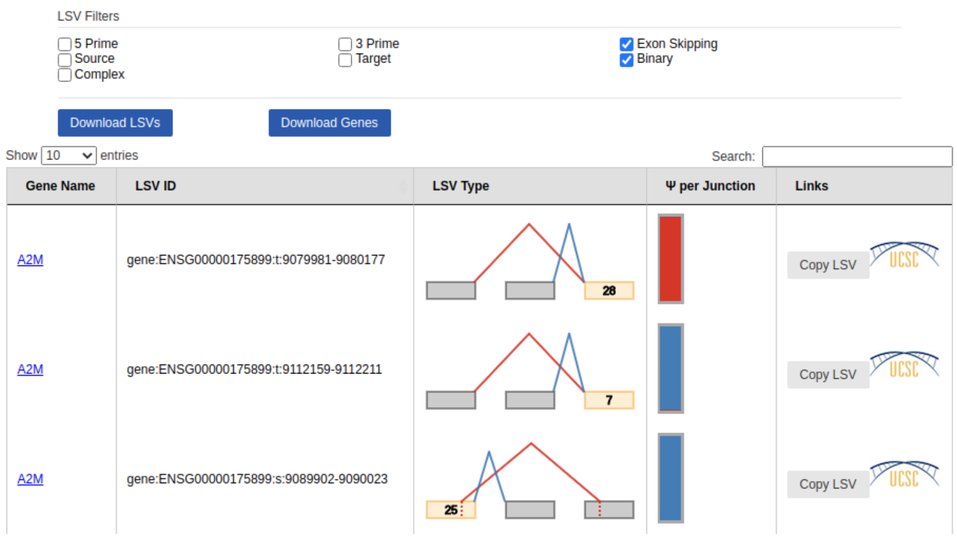


Figure 4.6: An example splice graph as displayed in VOILA [**majiq2**]. Exons, represented by grey rectangles, are connected with another exon via an edge, if a junction between the exons (or a variant of them) is observed. The number above an edge displays the number of observed raw reads spanning a particular exon-exon junction.

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) 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 [**galaxy**] from the European Nucleotide Archive (ENA). For each sample, the raw reads in FASTQ format were then mapped to the reference genome GRCh38 [**hg38**] using STAR [**star**]. 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.

Our implementation of PSI estimation was applied to the GTEx dataset. SUPPA and MAJIQ were applied to the data from the HipSci repository. We refer to the GTEx-based and with our own PSI estimation method processed dataset, to



LSV Filters

☐ 5 Prime
☐ Source
☐ Complex
☐ 3 Prime
☐ Target
☑ Exon Skipping
☑ Binary

Download LSVs
Download Genes

Show 10 entries
Search:

Gene Name	LSV ID	LSV Type	Ψ per Junction	Links
A2M	gene:ENSG00000175899:t:9079981-9080177	28		Copy LSV UCSC
A2M	gene:ENSG00000175899:t:9112159-9112211	7		Copy LSV UCSC
A2M	gene:ENSG00000175899:s:9089902-9090023	25		Copy LSV UCSC

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

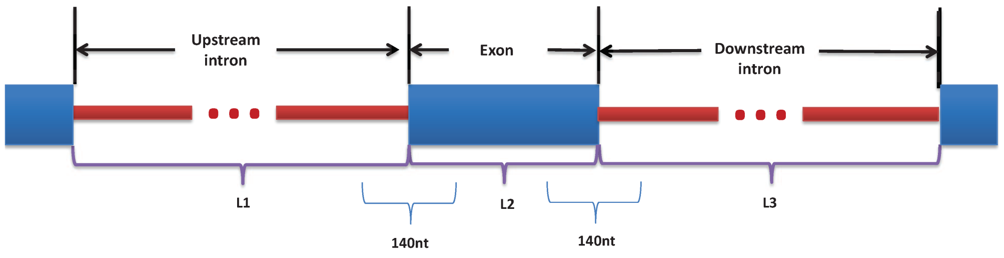


Figure 4.8: Schematic of model input for an exon-centric training sample [**dsc**]. The input are the 140 nucleotide region extracted around the exon start and exon end, as well as the normalized length of the exon and its flanking introns. The input for intron-centric training samples is analogue.

the HipSci-based and with SUPPA processed dataset and to the HipSci-based and with MAJIQ processed dataset as GTEx dataset, HipSci SUPPA dataset and HipSci MAJIQ dataset for brevity.

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-used exon or junction,
- start and end coordinates of the exon or junction within a chromosome,

- the lengths of neighbouring introns or exons*[1].

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 or regress its PSI. Intron-centric training samples are created when the task of the network is to classify a junction or regress its PSI.

Note that for intron-centric training samples we do not classify the intron or regress its PSI because the respective PSI is estimated with respect to the inclusion or exclusion of a particular junction. To estimate the PSI of an intron it would be necessary to also account for all other junctions which correspond to a particular intronic region being spliced in 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). The size of the nucleotide window was motivated by previous work [**dsc**].
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.
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 functional 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.

[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 0.

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 [**dsc**] [**flawed4**], 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 [**lengthsref1**] [**lengthsref2**]. 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 [**bestlengthsref**].

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 [**exonintronlens**] [**bionumbers**].

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 training label for the classification task is the scalar 1 if the respective exon or junction is constitutive, and 0 if not. The associated training label for the regression task is the PSI estimate.

4.3.3 Dataset statistics

Name	Tissue	Type	Misc	Number of samples
HEXEvent	mixed	exon	/	50,918
GTEx	brain cortex	exon	/	30,466
GTEx	cerebellum	exon	/	37,095
GTEx	heart	exon	/	19,257
GTEx	brain cortex	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	/	48,652
HipSci MAJIQ	iPSC	exon	diff lib	49,275
HipSci MAJIQ	iPSC	exon	diff indv	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. 'diff lib' abbreviates that the dataset was taken from the same individual, but a different library and 'diff indv' abbreviates that the dataset was taken from a different individual than the first shown HipSci MAJIQ iPSC cell-based dataset.

Table 4.1 shows the number of training samples in each dataset.Among the exon-centric datasets, the HEXEvent and HipSci MAJIQ datasets contain comparatively more training samples because they also contain constitutive non-cassette exons, whereas the other datasets don't. In contrast to the sensory neuron and iPSC cell-based HipSci datasets, the magnitudes of the GTEx-based datasets vary a lot between tissues: the cerebellum tissue dataset contains roughly twice as many training samples as the heart tissue dataset. On closer investigation, this is because the cerebellum tissue contains 10,000 genes whereas the heart sample only contains 5,000. Surprisingly, the heart tissue is based on 17 million exon-exon junction reads compared to 8 million exon-exon junction reads from the cerebellum tissue sample. 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.

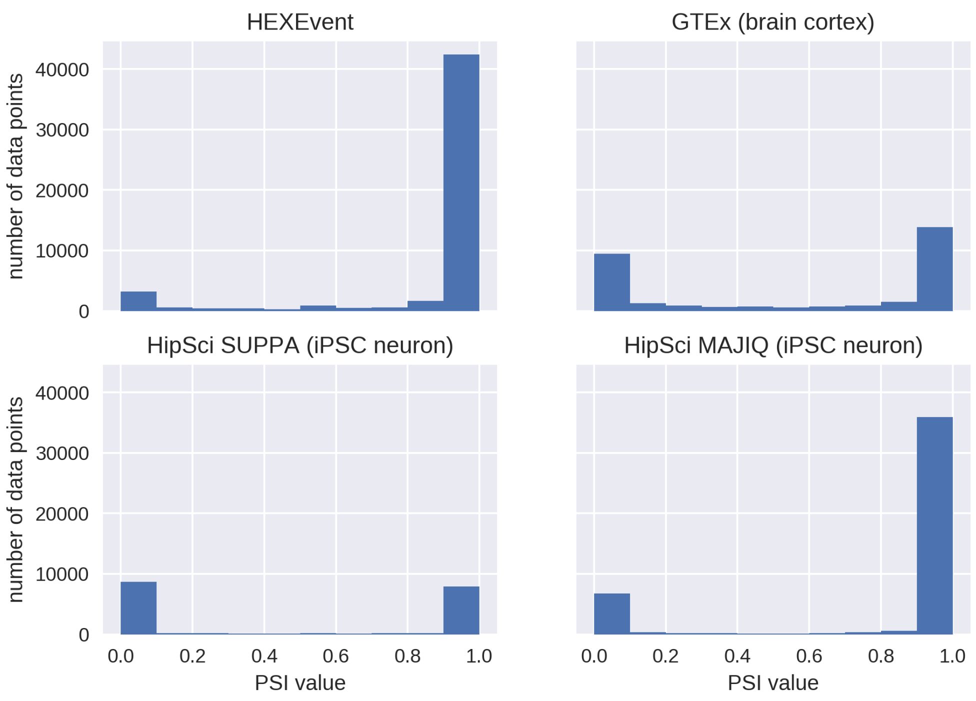


Figure 4.9: Histograms comparing the exon-centric versions of the four fundamentally different training datasets used. The shown GTEx dataset is based on samples from brain cortex tissue and the shown HipSci datasets are based on samples from iPSC cells differentiated to neurons.

The same cross-tissue observations also hold for the intron-centric GTEx-based datasets. In general, the intron-centric datasets contain roughly four times more training samples than the exon-centric datasets. This is because they do not only contain junctions of cassette exons (in which case they would contain roughly twice as many training samples as the exon-centric datasets), but contain all non-filtered junctions.

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 modes 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 included [**buschhertel**] [**bimodalpsivalues1**] [**bimodalpsivalues2**].

The lack of non-cassette constitutive exons is visible in the GTEx and HipSci SUPPA

datasets containing significantly fewer constitutive samples; the other two primary datasets contain roughly three times as many constitutive as non-constitutive samples.
In particular, the HipSci SUPPA 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 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 or regression. This component is implemented as a shallow multilayer perceptron (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 between 0 and 1.

4.4.1 DSC: CNN-based

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

The motivation behind using CNNs

DSC uses CNNs for feature extraction. CNNs are useful when the input data contains spatially invariant patterns. Since images are fundamentally made up

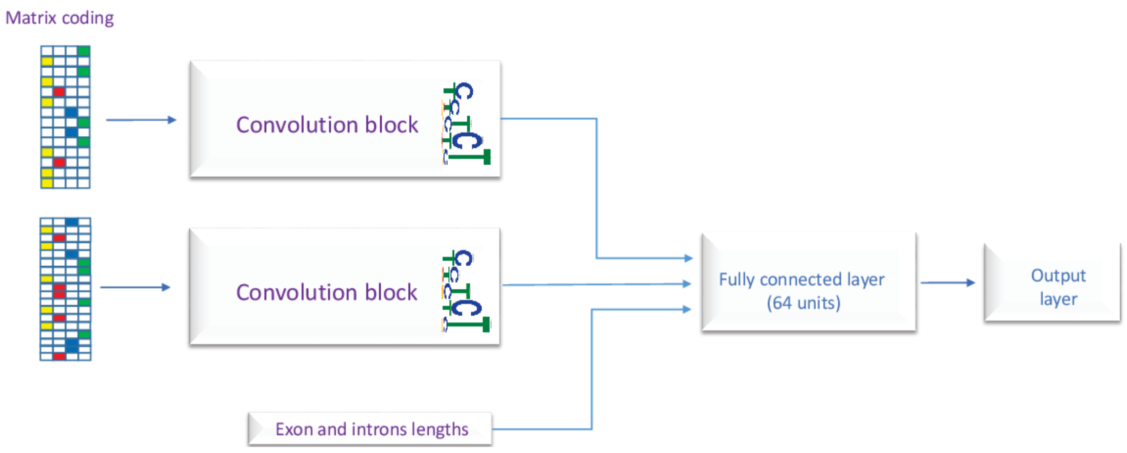


Figure 4.10: Architecture of DSC [**dsc**].

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.

Model architecture

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. The extracted features are concatenated with the lengths and used by a shallow MLP for predicting the output. The architecture of DSC is visualized in 4.10.

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 [**dropout**], a ReLU activation layer [**relu**] 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 [**dsc**]. 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 [**w2v1**] [**w2v2**], applying Recurrent Neural Networkss (RNNs) to text as in Sequence-to-Sequence (seq2seq) learning [**seq2seq**] and leveraging attention as in the extremely powerful and deep Transformer models [**allyouneed**] [**bert**]. 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 (this decision is discussed in more detail in Section 4.4.3). 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 Deep Learning innovations originating from NLP.

4.4.2 D2V: MLP-based

This model is a reimplementation of the model used for PSI regression on the mouse genome in [**d2vsplicing**]. The only architectural difference is that we possibly use a differently-sized MLP in the second part of the network, as architectural details on the MLP aren't given and no public implementation is available.

Introduction to word embeddings

Word2Vec is a neural network-based model which provides a continuously distributed (vector) representation for each word within a sentence [**w2v1**][**w2v2**]. 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: $\vec{King} - \vec{Man} + \vec{Woman} \sim= \vec{Queen}$ 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 also handle variable length texts (such as paragraphs and complete documents) [**d2v1**] [**d2v2**]. 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 [**ucsc**].
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 [**bionumbers**], we split the genome into sequences of 28,000 nucleotide sequences. Tests with splitting the genome into sequences of length 10,000 showed that the effect of this choice on model performance is imperceptible.

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 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 [**kmerlength1**] [**kmerlength2**], and found that the compromise of splitting each nucleotide sequence into overlapping 3-mers is a performant choice. Overlapping means that in the case of the sequence 'AACGAT' the resulting overlapping 3-mer sequence is 'AAC', 'ACG', 'CGA' and 'GAT'. Based on these findings, we split each sequence of nucleotides into overlapping 3-mers and obtain 27,998 overlapping 3-mers from the 28,000 nucleotides long pre-training sequences.

Pre-training of word embeddings

The pre-training for Word2Vec and Doc2Vec uses a shallow neural network with one hidden layer. 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 context 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. The context or sliding window around a word contains the words immediately adjacent to it. A typical window size is 5 words, meaning 5 words behind and after the current word will be used.

Doc2Vec works similarly. In the CBOW equivalent called distributed memory (DM), a document identifier (commonly simply 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 guideline 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 described above and using the DM training method, we trained a Doc2Vec model to output a 100-dimensional embedding for each document.

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 pre-training MLP[2]. In the case of 100-dimensional embeddings, this means that the hidden layer contains 100 neurons. 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

[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 of a MLP are represented as a $N \times d$ matrix, only the the d-dimensional column which belongs to the respective one-hot encoded word needs 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).

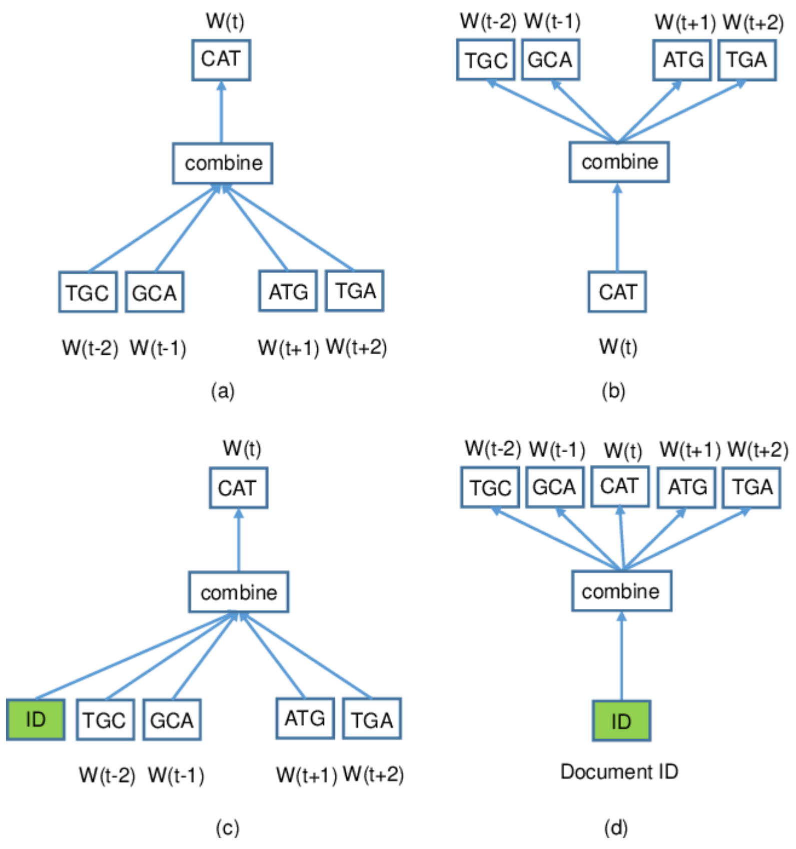


Figure 4.11: The four possible training algorithms for w2v and d2v models [**d2vsplicing**]: (a) CBOW, and (b) skip-gram for w2v, and (c) DM and (d) DBOW for d2v.

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) [**tsne**] 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 Alanine (Ala), Threonine (Thr), Proline (Pro) and Serine (Ser) in the top-left corner. Why are exactly these

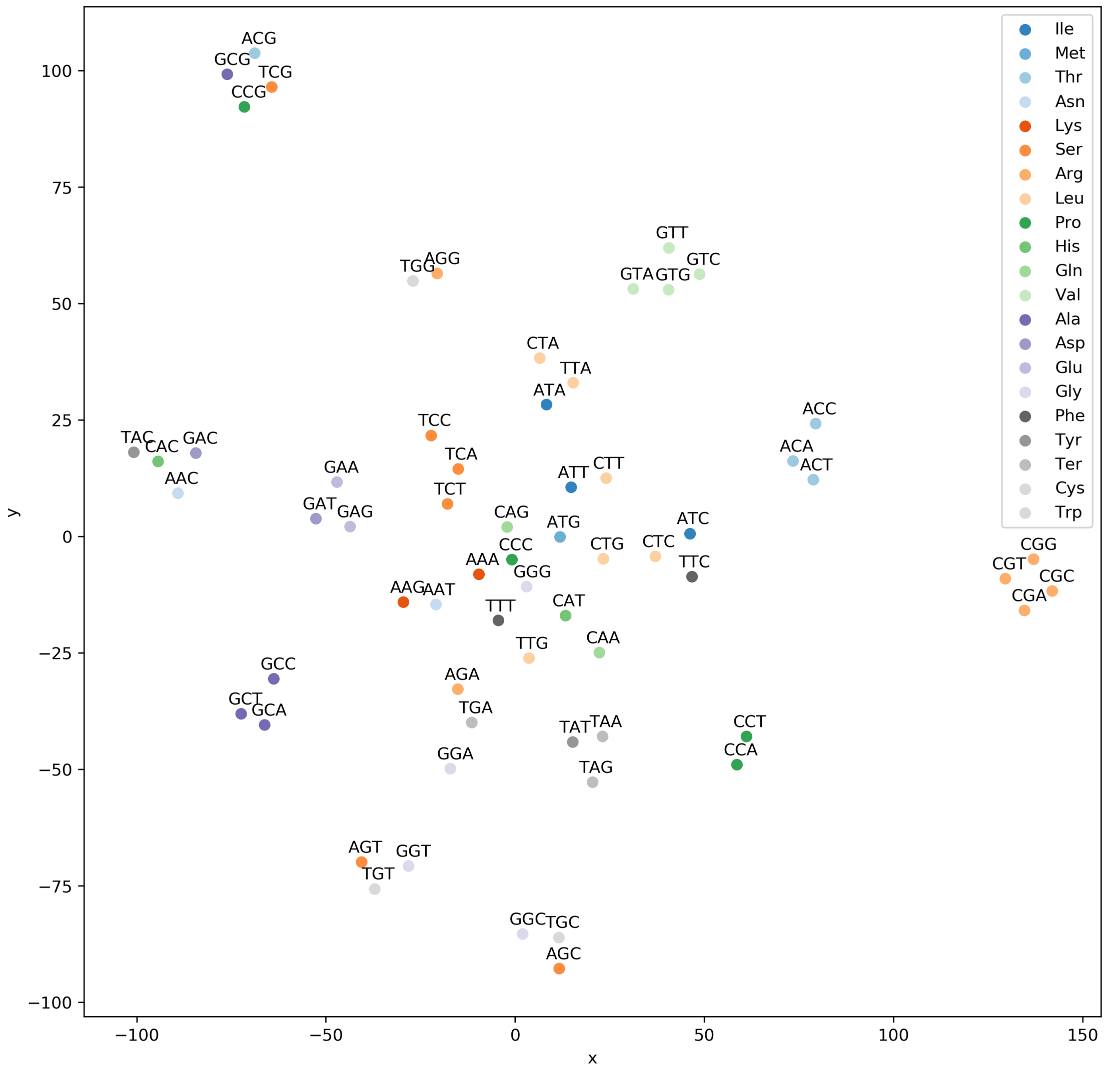


Figure 4.12: The 2-dimensional embeddings obtained by applying t-SNE to the 100-dimensional learned representations of all 64 possible 3-mers. The corresponding amino acids are color-coded.

different amino acids mapped together? The associated 3-mers reveal that these different amino acids share two nucleotides at the same position. This probably leads to these amino acids appearing in similar contexts. Overall, we take these visualizations as an indication that Doc2Vec was able to learn biologically useful embeddings for the overlapping 3-mers.

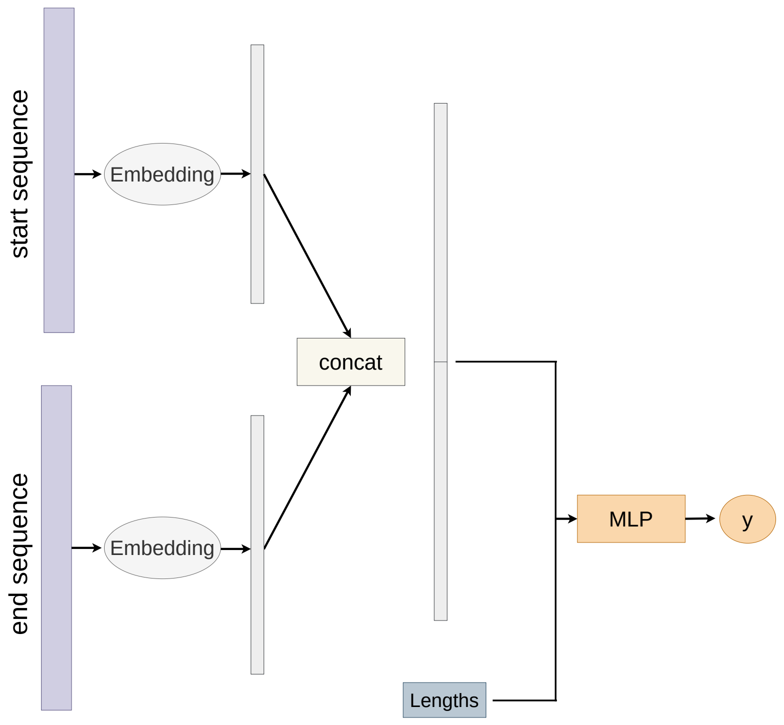


Figure 4.13: Architecture of D2V.

Putting it all together

The resulting model uses the pre-trained Doc2Vec network to obtain a 100-dimensional embedding of the extracted nucleotide sequences around the exon start and exon end. These two embeddings, along with the length features, are then concatenated and given to a shallow MLP to obtain the final prediction. We refer to this model as Doc2Vec (D2V) and visualize it in Figure 4.13.

4.4.3 RASC: BiLSTM and Attention-based

Seq2Seq

Sequence-to-sequence learning (Seq2seq) [**seq2seq**] 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 computed by the decoder. In the first timestep, the decoder uses the last output of the encoder and in

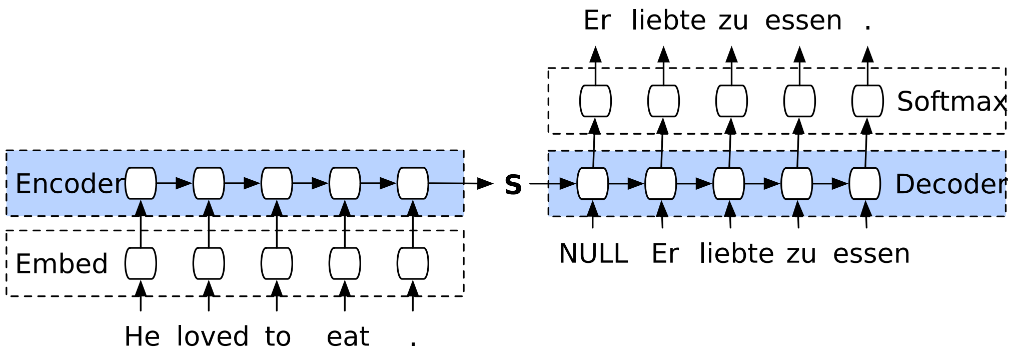


Figure 4.14: An encoder-decoder model [**ruderencdecgraphic**] translates the English sentence 'He loved to eat' to the German sentence 'Er liebte zu essen'. The decoder takes the last hidden cell state of the encoder as initial input and generates the output token-by-token. The decoder takes its own previous output as prompt for the next output and stops generating output token when it generates a '.'-token.

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 [**lstm**] or GRUs [**gru**]. Encoder and decoder are jointly trained to maximize the probability of a 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 [**googlemt2016**]. Figure 4.14 visualizes an encoder-decoder model.

Are you paying attention?

In the original Seq2seq framework, only the last state of the encoder is passed to the decoder. This means that the encoder needs to encode all the information observed in the input sequence into its last state. While theoretically possible, [**attention**] hypothesized that this informational bottleneck hampers performance in practice. They proposed removing this bottleneck via introducing the attention mechanism. Attention is a mechanism whereby the decoder can access all the representation generated by the encoder at once and can 'choose' to focus on (attend to) the most important ones. Introducing attention lead to large performance gains across Seq2seq-based models and quickly became standard practice [**attentionforms**]. 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.

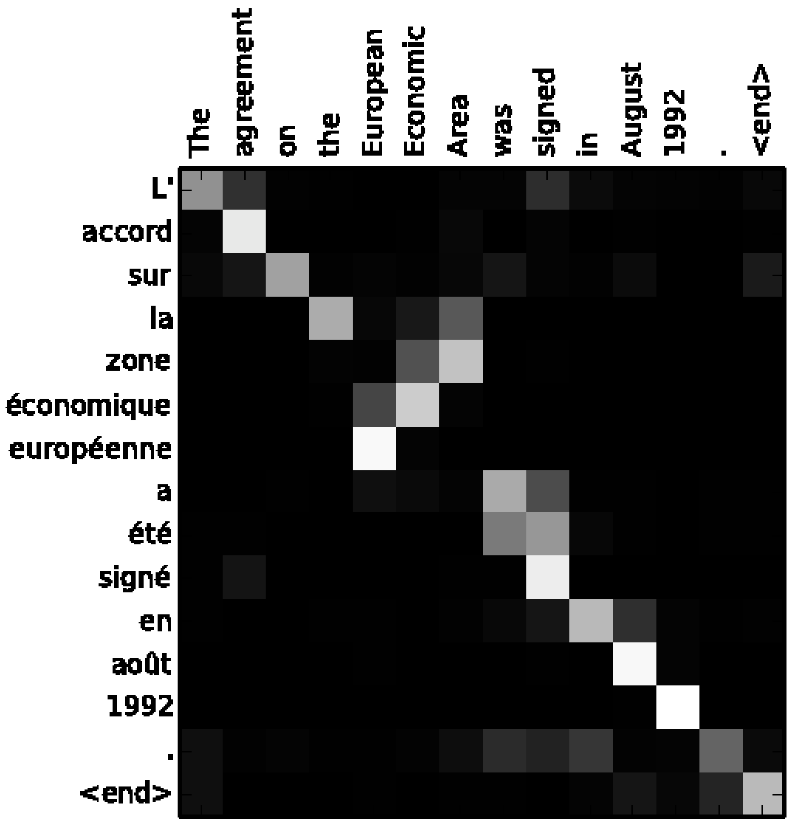


Figure 4.15: Attention heatmap when translating a sentence from French to English. A brighter color indicates that the model pays more attention to a particular word. The order of the constituent words in 'zone économique européen' and 'European Economic Area' is different between French and English, but the model learned to pay attention to the semantically corresponding word during translation.

Figure 4.15 visualizes what parts of an input sequence a model is paying attention to during a MT task. Attention is one of the most important innovations in Deep Learning research in recent years. It can theoretically be useful for any task where a model needs to make a decision based on certain parts of an input. For instance, the current state-of-the-art Transformer models [**allyouneed**] [**bert**] [**gpt3**] forego recurrent networks completely and solely use attention within the encoder and decoder components. It has also been successfully used in other domains such as recommender systems [**attention_recommender_systems**] or speech recognition [**attention_speech_recognition**].

Despite these successes, not many attention-based models have been applied to genomic data yet and in particular, none have been applied to the prediction of splicing behaviour.

Why LSTM-based models were chosen over Transformers

Perhaps surprisingly, we chose to not evaluate Transformer models for this task: The main advantage of Transformers over LSTMs is their parallelizable nature enabling them to scale to enormous datasets and hundreds of millions (or billions [**gpt3**]) of parameters [**allyouneed**]. This is because Transformers can process all inputs in a sequence concurrently, in contrast to LSTMs who need to process them sequentially. However, there are no huge datasets to scale to: our largest exon dataset contains roughly 50,000 training samples and this already includes all the known cassette exon in the human genome. Creative use of data augmentation might increase dataset size by two or three factors, but even 150,000 training samples is still far smaller than the millions of training samples used in NLP. Scaling down Transformers models would mean they lose their main advantage over LSTMs: on small NLP datasets LSTMs and small (not-distilled [**distilbert**]) Transformers perform similarly [**mogrifier**][**smalltransformerwin**] and there is no clear consensus which approach is superior.

Additionally, the autoregressive bias inherent to LSTMs is likely important for interpreting genomic sequences: . Transformers are stateless and only add information about the relative positions of the input through the use of positional encodings. This would mean that the learning task of the Transformers is harder. Thus, we base the encoder part of our model on the simpler LSTMs.

We now describe the modifications we made to the Seq2Seq and attention set-up known from MT for alternative splicing classification and regression. 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. Thus, we don't require a recurrent decoder and instead use a shallow MLP in its place.

Modification 2: Attention with a learned query vector

The most common form of attention (multiplicative or dot-product self-attention [**allyouneed**]), makes uses of conceptual queries and key-value pairs. A query, key and value matrix is assigned to each symbol in the input sequence. 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 more formally:

Initially, 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:

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

where $I \in \mathbb{R}^{l \times in}$, and $W^Q, W^K, W^V \in \mathbb{R}^{in \times out}$, 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.

Attention can be then computed as:

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

where $Z \in \mathbb{R}^{l \times out}$. Note that this makes use of the fact that $Q \cdot K = QK^T$ where $\cdot$ 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 (finding a good representation is the task of the encoder), 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. The input I is a concatenation (along the first dimension) of the representations given by the encoder for the nucleotide sequence around the exon start and exon end.

4.4.4 Further modifications of the attention mechanism

In the following, we introduce three further modifications of the attention mechanism which we evaluated separately from the modifications described so far.

No query matrix

As the query matrix Q^* is the same independent of the exact input sequences, it could be possible to remove the query matrix and compute the attention weights sorely based on the key matrix K. This is possible by learning a key weight matrix $W^{K_{no\_query}} \in \mathcal{R}^{in \times 1}$ and computing attention as:

$$K_{no\_query} = IW^{K_{no\_query}}$$

$$Z_{no\_query} = attention^{**}(I) = softmax(K^*)V$$

where $K_{no\_query} \in \mathbb{R}^{l \times 1}$ and $Z_{no\_query} \in \mathbb{R}^{1 \times out}$. This has the advantage of reducing the number of learnable weights as adapted the key weight matrix $W^{K_{no\_query}} \in \mathcal{R}^{in \times 1}$ is smaller than the initial key weight matrix $W^K \in \mathcal{R}^{in \times l}$.

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 [**allyouneed**] via h different sets of query, key and weight matrices $W^{Q^*_1}, W^{K_1}, W^{V_1}, \dots W^{Q^*_h}, W^{K_h}, W^{V_h}$ producing h outputs $Z_1, \dots, Z_h$. Applying this extension in our context engenders the following formulas:

$$K_i, V_i = IW^K_i, IW^V_i$$

$$Z^*_i = attention^*_i(I) = softmax(Q^*_i K^T_i)V_i$$

Note that the query weight matrices $W^{Q^*_1}, \dots, W^{Q^*_h}$ are again directly learned, like in the single head case.

The final output matrix $Z_{final} \in \mathcal{R}^{l \times out}$ is then computed as:

$$Z_{final} = concat(Z^*_1, \dots, Z^*_h)W^O$$

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 h attention heads.

While these extra representational subspaces might make the model more powerful and also more robust (in the sense that multiple attention heads act like an ensemble model), it also increases the number of learnable weights potentially leading to overfitting.

Convolution in attention heads

The information contained in a single nucleotide is very low. This motivated [**ghentransformers**] to integrate an additional 1D convolution into the computation of the query, key and value matrices Q, K and V which is applied after the multiplication with the weight matrices W^Q, W^K and W^V. This convolution allows the model to more effectively integrate information from multiple neighbouring nucleotides. This extension can also be understood as an alternative to splitting the input nucleotide sequence into k-mers (similar to the split into overlapping 3-mers for the input of the D2V model 4.4.2) and provided better results than splitting the input nucleotide sequence into k-mers when using a Transformer network for a genome annotation task in [**ghentransformers**]. In formulas, we can describe the adapted attention mechanism as:

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

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

where K and V are both fed through the same convolutional layer to arrive at K_{conv} and V_{conv}. The weight matrix corresponding to the convolutional layer is chosen so it has the same number of feature dimensions as K and V, that is, $W_conv \in \mathbb{R}^{out \times kernel \times out}$. $kernel$ denotes the size of the convolutional kernel. The padding is chosen so that the first dimension of K and V stays the same which leads to $K_{conv}, V_{conv} \in \mathbb{R}^{l \times out}$.

To incorporate the domain knowledge that the input sequences to our model come from two different places in the genome, the convolution is applied in such a way that it does not mix the information between the sequences around the exon start and around the exon end. To achieve this, we implemented the following computations:

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

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

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

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 implemented for V_{conv}.

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 [**annotatedtransformer**] and so we follow suit. Thus, to be precise, the query, key and value matrix computation should be:

$$K, V = IW^K + B^K, IW^V + B^V$$

$$Z^* = attention^*(I) = softmax((Q^*K^T) + B^{Q^*})V$$

where the query, key and value bias matrices B^{Q^*}, B^K and $B^V \in \mathbb{R}^{1\times out}$.

Putting it all together:

Like in a Seq2Seq model, an RNN-based Encoder is used to capture a representation of the input sequence. We attend to the most important representations of the encoded input using the modified attention mechanism we introduced. The resulting feature vector is concatenated with the lengths and given to a shallow MLP.

The input to the encoder blocks is a dense 4-dimensional embedding of the one-hot encoded sequences. This dense embedding is the same for each sequence and jointly learned during training. The extra embedding layer is included as previous work [**embeddingneeded**] has shown that otherwise the model might suffer from limited generalization performance, caused by the sparsity of the one-hot encoded representations.

LSTM cells opposed to unmodified RNNs are used as encoder as LSTMs have been shown to be better at retaining information over long sequences and avoiding the vanishing and exploding gradient problems [**lstm**]. In particular, we use Bidirectional LSTM (BiLSTM) [**bidirection**] to model that the splicing of an exon likely depends on bidirectionally interacting factors. A separate BiLSTM is used for finding for encoding the exon start and exon end sequence.

As this lead to empirically better results, we also introduce a 1D batch normalization layer [**batchnormalization**] before the attention layer. A dropout layer in the attention head is used for the same reason.

We pre-empt the results chapter by noting that we only use the extension of multiple attention heads of the three additional modifications of the attention mechanisms we considered. The architecture of our model is visualized in Figure 4.16. We refer to this new model as Recursive Attentive Splicing Code (RASC). In the experiments section, we validate the choice of introducing the attention mechanism to RASC, by removing it and evaluating the model which only uses the hidden state of the BiLSTMs. We denote this model as Recursive Splicing Code (RSC).

For RASC, 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 a grid search on the HipSci MAJIQ dataset.

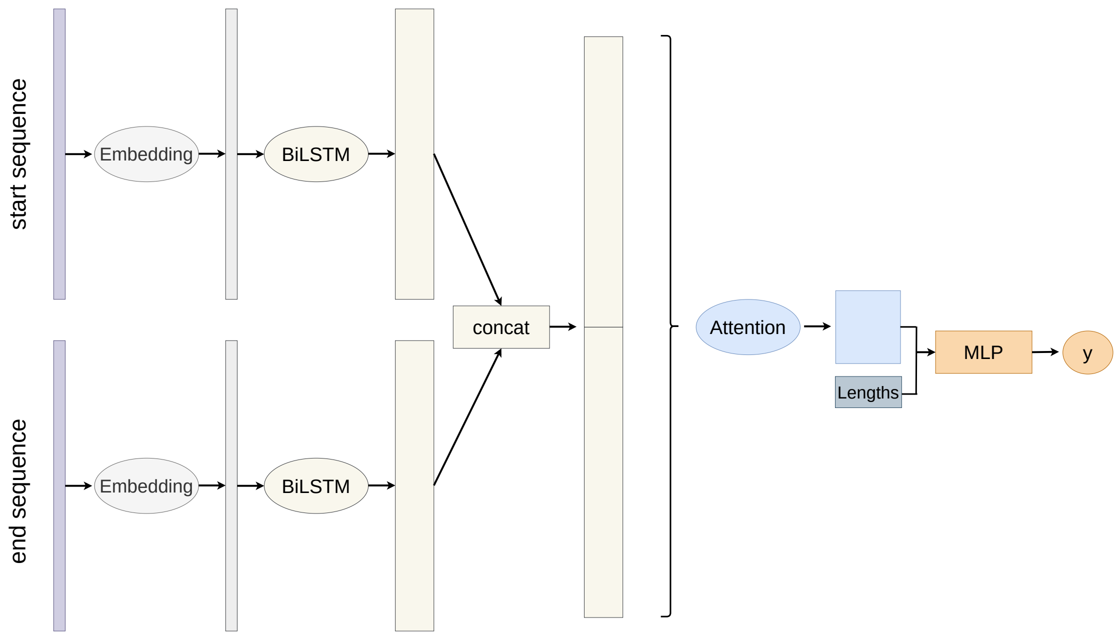


Figure 4.16: Architecture of RASC.

4.5 Implementation, training and evaluation

4.5.1 Implementation

All models, except the Doc2Vec network, were implemented using the PyTorch library (version 1.5.0) [**pytorch**]. The Doc2Vec model was implemented using the gensim library (version 3.8.3) [**gensim**]. PyTorch and gensim are both standard choices when implementing Deep Learning algorithms or text embedding models. 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 (version 2.3) and MAJIQ (version 2.0) 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 10 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 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 base classes were then 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 a separate JSON file. 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 give more details about the code structure, how to run experiments and what naming conventions are used.

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 networks we evaluate are all relatively shallow. For training and testing, 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 and we compute the mean test set performance. 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.

The pre-training of the Doc2Vec model on the human genome took 3 hours.

DSC, D2V and RASC 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). The increased training time of RASC is likely also influenced by LSTMs processing the input sequentially.

The hyperparameters used for training the models are given in Table 4.2. More details on the Doc2Vec pre-training procedure and hyperparameters values are given in the appendix 7.2.

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 loss is the standard loss for training Deep Learning-based (binary) classifiers.

4.5.3 Evaluation

To evaluate the models, we report the mean test set performance along with standard deviation across (cross-validation) runs. Where we established in a base experiment that the variance for a given model was low between runs (lower than 1%), we only trained the model once in follow-up experiments for practical reasons.

Previous studies have shown that model performance varies significantly between lowly and highly included alternatively spliced exons [**dsc**] [**buschhertel**]. Thus, following previous work, we split our test set into a set of highly (PSI > 80%) and a set of lowly to moderately included, alternatively spliced exons (PSI <=80%). Both sets contain all constitutive exons. We denote the first set as high PSI dataset (HighPSI), the second as low PSI dataset (LowPSI) and the complete test as all PSI dataset (AllPSI). We evaluate our models on all three datasets.

Metrics

As in previous work [**dsc**] [**buschhertel**], we use the Area under the receiver operating characteristic curve (AUC) as evaluation metric. The AUC provides an aggregate measure of the classification performance across all possible decision thresholds.

However, the AUC doesn't provide information about model performance at a particular decision threshold. To evaluate whether a model is fit for clinical use, decision threshold-specific metrics like sensitivity and specificity are crucial to know. To evaluate these, we analytically compute the decision threshold which maximizes the F1 score. The F1 score provides an aggregate measure which treats precision and recall equally and is defined as the harmonic mean of precision and recall. It is a common measure to evaluate the performance of classifiers and we use it as starting point for our further analysis.

Again following previous work [**d2vsplicing**] [**jha**], we use the explained variance R^2 to evaluate model performance on the regression task.

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
RASC	
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 4.2: Hyperparameters of the three models

5 Results

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

Results are given as obtained on the test set. Across all datasets and experiments, validation and test set metrics differ by less than 1% on average. We observe no undue overfitting and unless explicitly mentioned, AUCs on the training set are generally higher by 0.02 to 0.06.

5.1 HEXEvent dataset

To obtain a baseline, we evaluate all three models on the HEXEvent dataset as used in [**dsc**] 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%. While differences are small, RASC seems to perform slightly better than the other models.

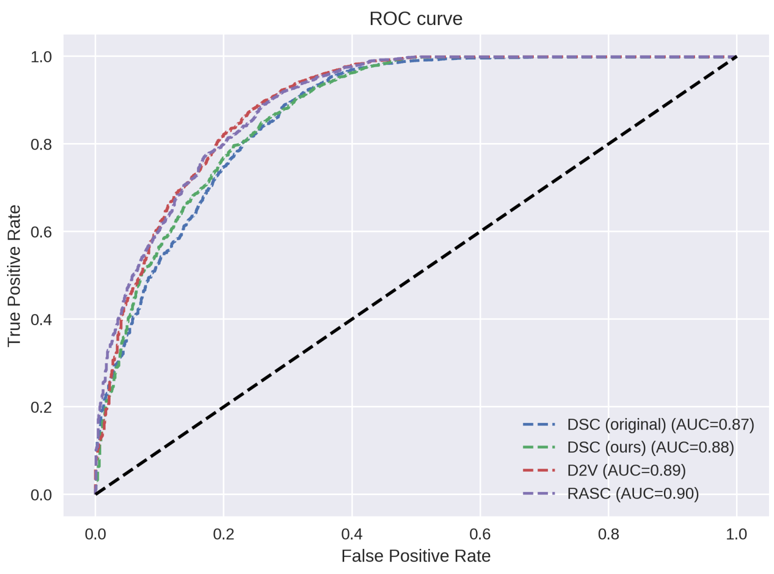


Figure 5.1: Comparison of the ROC curves of the three models as well as the original implementation on the HEXEvent dataset. The values of the original implementation were obtained by rerunning the training on the publicly available implementation.

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

Length features are necessary

Reimplementation was completed piece-wise, that is, first the sequence features were added as input, then the networks were trained to ensure 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

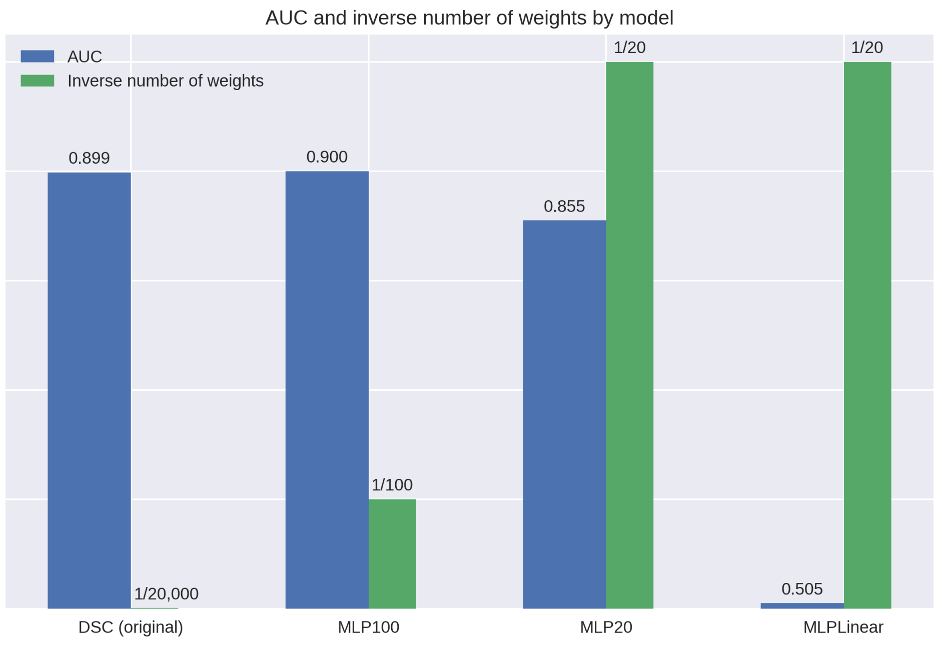


Figure 5.2: Stress testing the HEXEvent dataset used in [**dsc**]. The graph shows the performance as well as the inverse of the respective model sizes used.

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 $\frac{AUC - AUC_{no\_lengths}}{AUC - 0.5}$. 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 MLPs with one hidden layer which only take the length features as inputs. MLPLinear doesn't contain 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 [**dsc**] with these very simple models using two to three orders of magnitude fewer parameters (note that DSC has 20,001 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 data 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.
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 ongoing for over 40 years.
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 [**dsc**]. 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 data leakage bug, the performance of the linear model likely wouldn't break down either. Therefore, we judge bugs in our reimplementation 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 [**dsc**]'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 in passing, multiple papers use a HEXEvent-derived dataset for the training of Machine Learning models such as SVMs [**buschhertel**], Random Forests [**flawed4**]

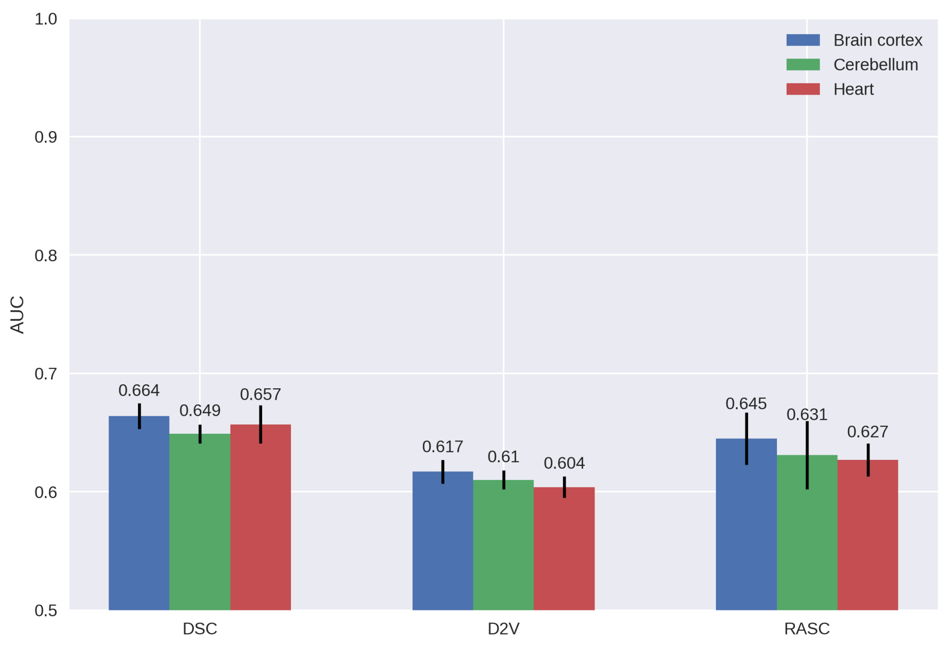


Figure 5.3: Performance on the GTEx-based exon-centric datasets across different tissues. The error bars give the standard deviation across all cross-validation runs.

[**flawed1**], Decision Trees [**flawed2**] or AdaBoost-based algorithms [**flawed3**]. The validity of these papers results' are called into question by these findings.

These data quality issues motivated us to construct alternative, better, datasets. We now give their results.

5.2 GTEx-based datasets

5.2.1 Exon-centric datasets

Results are given in Figure 5.3. Across all models, performance is poor and the predictive power of the models is low. Surprisingly, performance is also very similar across tissues: the mean cross-tissue performance difference of 0.007 AUC is smaller than the mean cross-run difference of 0.014 AUC. This is surprising from a biological perspective as cross-tissue splicing differences are well-documented [**crosstissuesplicing**]. This could indicate that, while splicing across tissues varies, the relative complexity of cassette exon splicing behaviour across tissues is similar. However, this would be overinterpreting models with low predictive power. Most

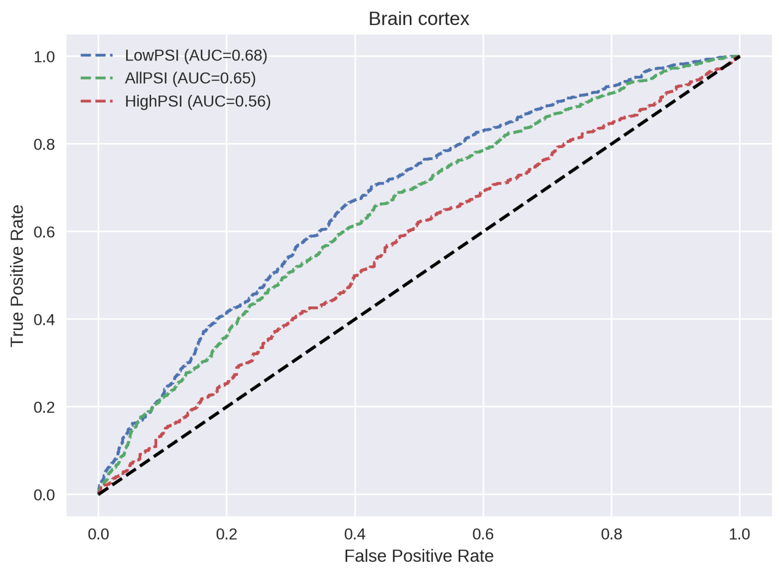


Figure 5.4: ROC curve of RASC on GTEx-based exon-centric brain cortex dataset.

likely, the models are already struggling to learn the baseline splicing behaviour invariant across tissues and don't learn finer cross-tissue differences. The low cross-tissue performance differences also surprising from a machine learning perspective, as the cerebellum-based dataset contains almost twice as many training samples as the heart-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 significantly more training samples. All models perform best on the dataset based on a brain cortex sample.

Relative model performance stays the same between tissues: DSC tends to perform best, followed by RASC and the D2V model. The variance of RASC between runs is on average twice as twice as high as the respective variance of DSC and D2V. As a result, RASC 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 RASC needed more regularization on this dataset and dataset specific fine-tuning could've lead to it performing the best across cross-validation runs.

Figure 5.4 gives more insight into 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 $< 80\%$). Per AUC definition, the AUC on the allPSI dataset 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 mirror analogue observations made on the HEXEvent dataset [**dsc**].

5.2.2 Intron-centric datasets

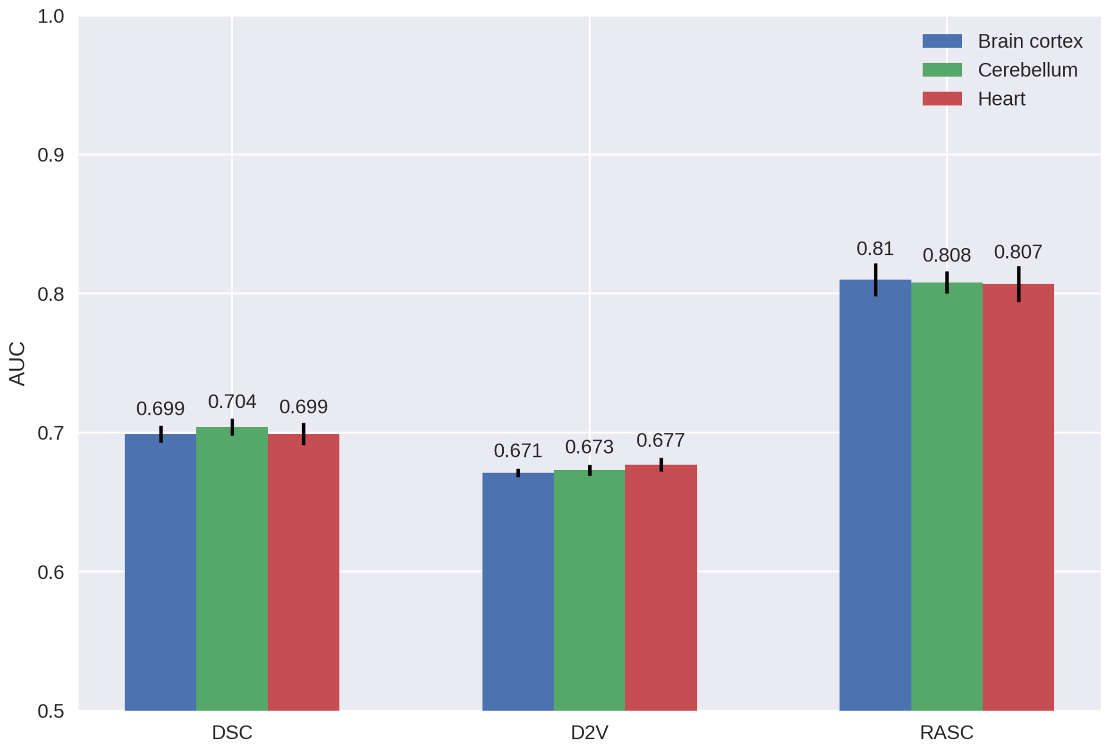


Figure 5.5: Performance on the GTEx-based junction datasets across different tissues.

Main analysis

Figure 5.5 shows that junction classification is seemingly an easier task: the AUC of all models increases by at least 0.05 to 0.06 compared to the exon classification task. However, this is likely a result of the intron-centric datasets containing on average four times more training samples rather than due to different inherent task difficulties. RASC significantly outperforms DSC and D2V.

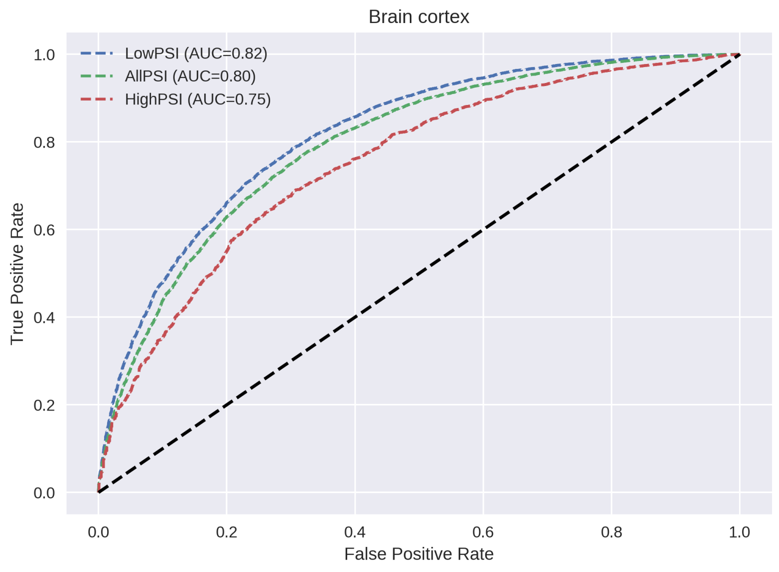


Figure 5.6: ROC curve from training RASC on the GTEx-based intron-centric heart-tissue-based dataset.

Similar to the exon-centric datasets 5.2.1, we observe very low cross-tissue performance differences. We again believe that model performance is generally too poor to warrant speculation about the underlying biological realities. However, we come to a different conclusion regarding the observation that the different dataset sizes between tissues don't affect model performance: for the GTEx-based exon-centric datasets we hypothesized that the models have hit diminishing returns for adding more data or that they need significantly more data. As the number of training samples has quadrupled in the move from exon- to intron-centric datasets, we now believe that the models have hit a point of diminishing returns for adding more data.

The performance across rarely and highly included junctions is less disparate than on the exon-centric datasets (see Figure 5.6). This indicates that the models are better able to learn to recognize motifs which separate constitutive and highly alternatively spliced junctions than for exons.

Assessing model performance without length features

As model performance is more promising than on the GTEx-based exon-centric datasets, we validate that this isn't due to an overreliance on the length features

Model name	Only sequences	Sequences + lengths	Relative performance improvement
DSC	0.661	0.704	0.211
D2V	0.629	0.673	0.254
RASC	**0.776**	**0.808**	0.104

Table 5.2: Performance on the GTEx-based intron-centric dataset with and without length features.

as on the HEXEvent dataset. Table 5.2 shows that the length features still make up a large part of model performance, but that the models don't rely on them completely. However, the performance proportion the length features account for is still biologically implausible. Thus we conclude that the dataset is an improvement over the HEXEvent dataset, but generating a dataset where model performance is less dependent on the length features is still desirable.

Evaluation of the GTEx-based datasets

Overall, we observe poor to promising model performance, biologically surprising low cross-tissue performance differences and biologically implausible reliance on the length features. In 4.3.1, we mentioned issues when trying to estimate PSI naively. Although we alleviated these in pre-processing, we don't account for the information contained in non-junction reads and don't integrate information from multiple samples. As a result, data quality is likely still an issue distorting the results in this section. Thus we conclude that the GTEx-based datasets improve upon the HEXEvent dataset, but using datasets based on more accurate PSI estimation methods is desirable and would lead to more meaningful results.

5.3 HipSci SUPPA datasets

5.3.1 Dataset based on sensory neuron cell lines

Main analysis

Figure 5.7 shows how the model perform very poorly on the HipSci SUPPA dataset. The performance is worse than on the GTEx-based exon-centric dataset

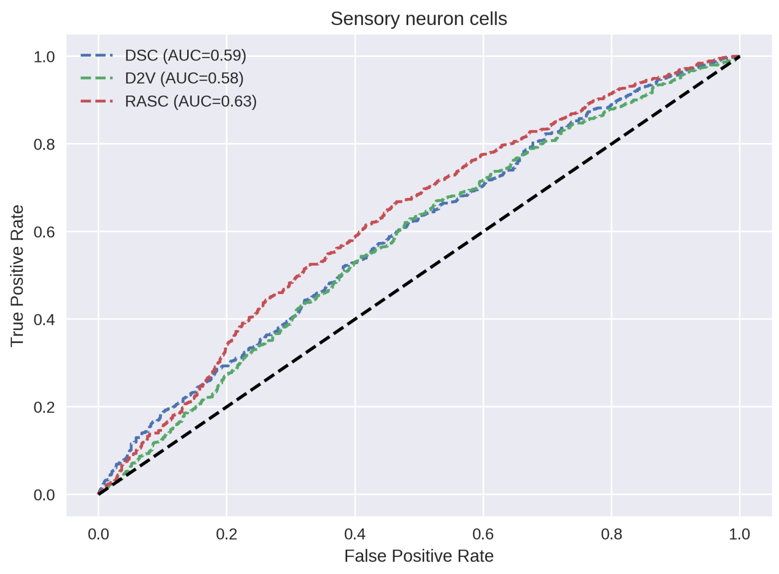


Figure 5.7: Comparison of the ROC curves of the three models on the HipSci SUPPA dataset. The dataset is based on iPSC cell lines differentiated to sensory neuron cell lines.

and roughly equal to the performance on the HEXEvent dataset without length features. This indicates that

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 due to SUPPA not addressing various challenges in PSI estimation and due to SUPPA not generating constitutive training samples (see also 4.3.1). SUPPA does not appear to be accurate enough for our purposes. Therefore, we don't evaluate our models on the HipSci SUPPA dataset based on undifferentiated iPSC cell lines.

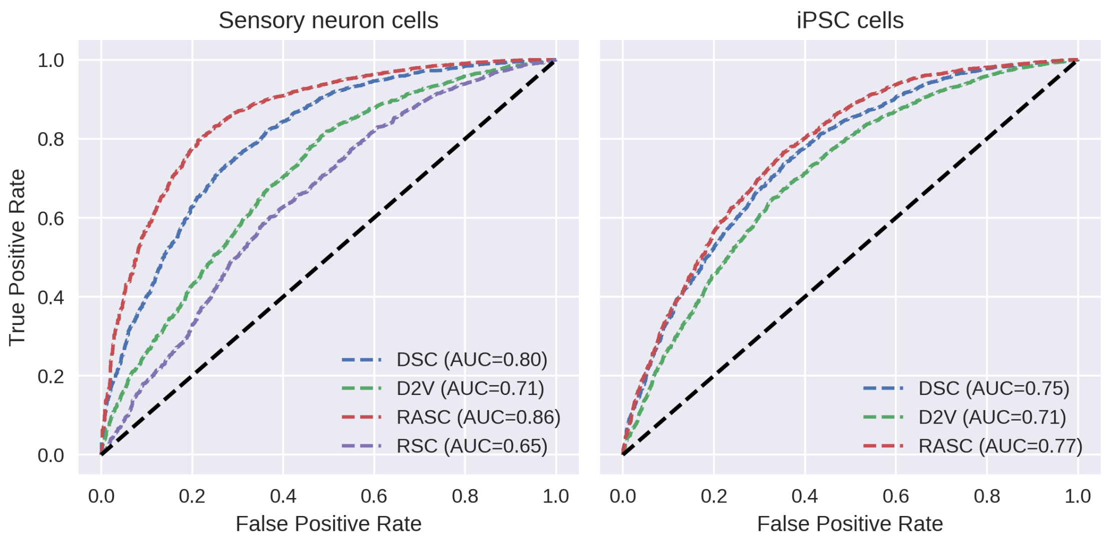


Figure 5.8: Left: ROC curves on HipSCi MAJIQ dataset derived from iPSC cells differentiated to neurons. Right: ROC curves on HipSCi MAJIQ dataset derived from undifferentiated iPSC cells.

5.4 HipSci MAJIQ datasets

5.4.1 Dataset based on sensory neuron and iPSC cell lines

Main analysis

All models perform significantly better than on previous datasets and we observe stark disparities between the models (see Figure 5.8). While DSC outperforms D2V by a significant margin, it is outperformed by a similar margin by RASC. Adding attention to RSC promotes it from the worst to the best performing model, validating our choice. Performance on the sensory neuron cell dataset is generally higher by 2%, indicating that the captured splicing behaviour is easier to predict for the models.

Impact of architectural choices

The poor performance of D2V is likely a result of its architecture: the 100-dimensional embeddings used for classification weren't optimizely jointly during training and likely don't contain fine-grained information, such as the specific positons of motifs, required for accurate splicing prediction. We conjecture that D2V would benefit from receiving further split input sequences, e.g. input sequences containing 35 nucleotides. The best solution, while keeping the word embeddings, would likely be to embed the overlapping 3-mers in the sequence via word2vec and combine this with an efficient convolutional architecture, as done in [**d2vsplicing**].

Assessing the dataset quality

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

Table 5.3: Performance on the dataset based on processing with MAJIQ and iPSC cells differentiated to neurons with and without length features given.

As litmus test for dataset quality, we repeat the experiment of removing the length features as model input. We document the results in Table 5.3. They reveal that a large part of D2V's performance relies on information contained in the length features. However, since removing the length features only marginally affects the performance of DSC and RASC, we conclude that this observation is symptomatic of D2V being unable to effectively incorporate information from the sequences (see discussion in 5.4.1). rather than that the dataset is not biased in the same way as the HEXEvent dataset.

Giving the strong model performance independent of the length features and the prior expectation for MAJIQ to produce the highest quality PSI estimates, we conclude that it does. This indicates that the comparatively poor performance on the GTEx-based and HipSci SUPPA datasets are due to poor data quality, and not due to the inherent difficulty of the splicing classification task. Thus, using the HipSci MAJIQ datasets is preferable to using any of the other datasets we have evaluated.

Motivated by this observation, we use the HipSci MAJIQ dataset to validate our architectural choices for RASC and to carry out further analysis omitted on the other datasets.

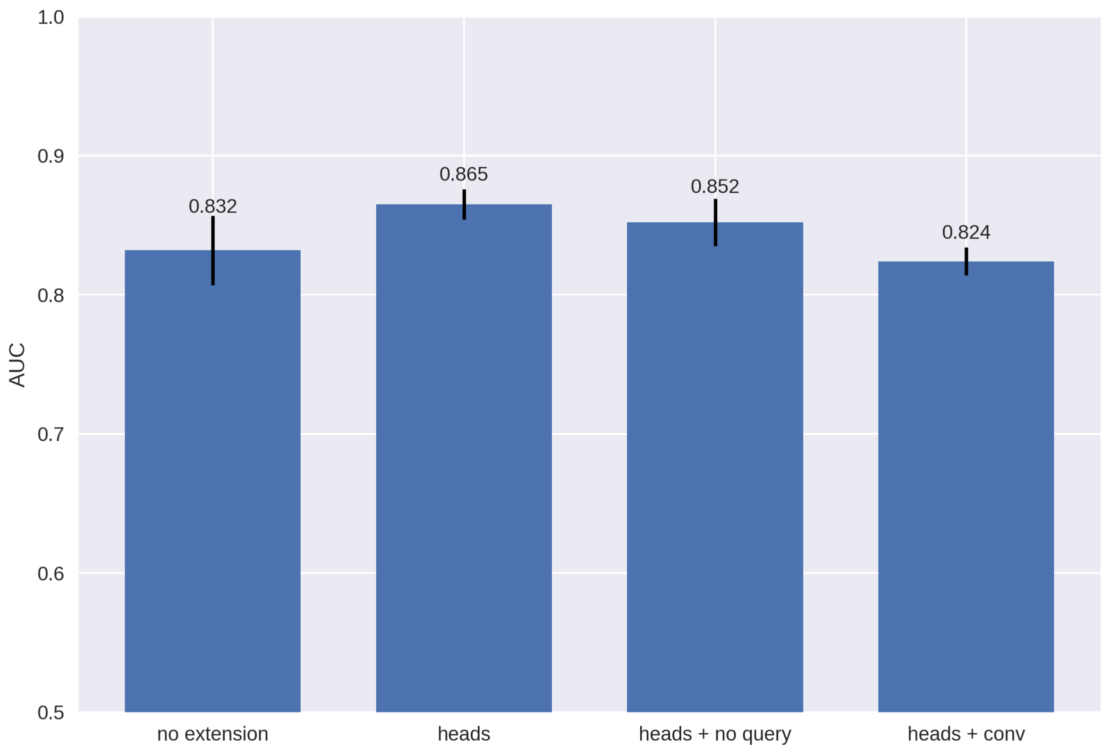


Figure 5.9: Performance evaluation of the three extensions to the attention mechanism. We abbreviate the use of multiple attention heads as 'heads', the removal of the query matrix as 'no query' and the introduction of an additional convolution as 'conv'.

5.4.2 Evaluation of the additional attention modifications

We evaluate the three additional attention modifications described in 4.4.4 by testing them sequentially. The results of these tests are shown in Figure 5.9.

Using multiple attention heads

Using multiple attention heads leads to significantly improved performance and to significantly reduced variance between runs. Thus, the additional representational subspaces appear to improve performance and the ensemble model effect of multiple attention heads appear to reduce variance. No overfitting is observed, despite the number of model weights increasing from roughly 36,000 to 67,000 compared to the single attention head model. Due to the uniformly improved performance, we elect to incorporate multiple attention heads into RASC.

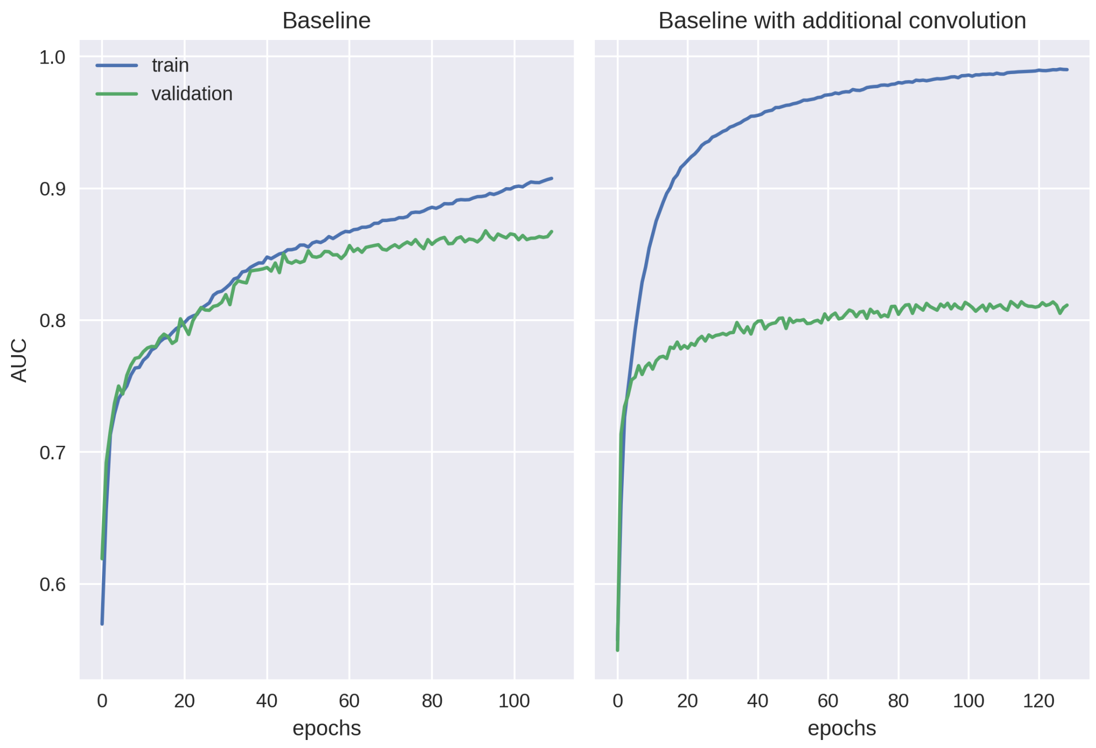


Figure 5.10: RASC overfits when the additional convolution operation is introduced into the attention heads. The baseline refers to RASC with multiple attention heads.

Removing the query matrix

We observe slightly diminished performance upon removing the query matrix. The extra representational capacity afforded to RASC by having query matrices appears to improve its predictive power. Despite decreasing model size by roughly 10,000 parameters when removing the query matrices, we decide to not incorporate this adaption of the attention mechanism.

Adding an extra convolution

Adding an additional convolution operation to the attention mechanism, surprisingly leads to decreased performance, even comparing unfavorably to the baseline single-head attention model. This in contrast to [**ghentransformers**] where this extension lead to very significant relative performance improvements. There are likely multiple factors contributing to these varying results:

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, the model can already encode information about the neighbouring nucleotides 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-nucleotide resolution. Thus, the convolutional layers likely give the model less additional modelling flexibility.
2. In [**ghentransformers**], the task is full genome transcription start site annotation (so predicting where genes start). Therefore, their task occurs at the inter-gene level while our task occurs at the intra-gene level. This likely means that the motives [**ghentransformers**]'s model has to consider generally span more nucleotides and granting the model more capacity to integrate information over multiple nucleotides is more crucial. In contrast, granting RASC more capacity to integrate information over multiple nucleotides may not be necessary to identify the shorter motifs influencing splicing.
3. Furthermore, we observe overfitting when using this extension. The training curve in Figure 5.10 showcases this behaviour. We observe overfitting despite using the same convolution across multiple heads, increasing the dropout probability in the attention heads, introducing additional batch normalization layers and limiting the convolution kernel size to 3 .

In summary, we extend RASC with multiple attention heads. The results of RASC we report on the other datasets also use the model with multiple attention heads.

5.4.3 Sensitivity and specificity analysis

We present the confusion matrix for the binary classifier based on RASC in Figure 5.11. The decision threshold which maximizes the F1 score was chosen resulting in a F1 score of 0.88 (for the negative class of constitutive exons). We observe comparatively few false positives, but many false negatives and correspondingly obtain a low sensitivity score of 0.56, but a high specificity score of 0.93. Thus,

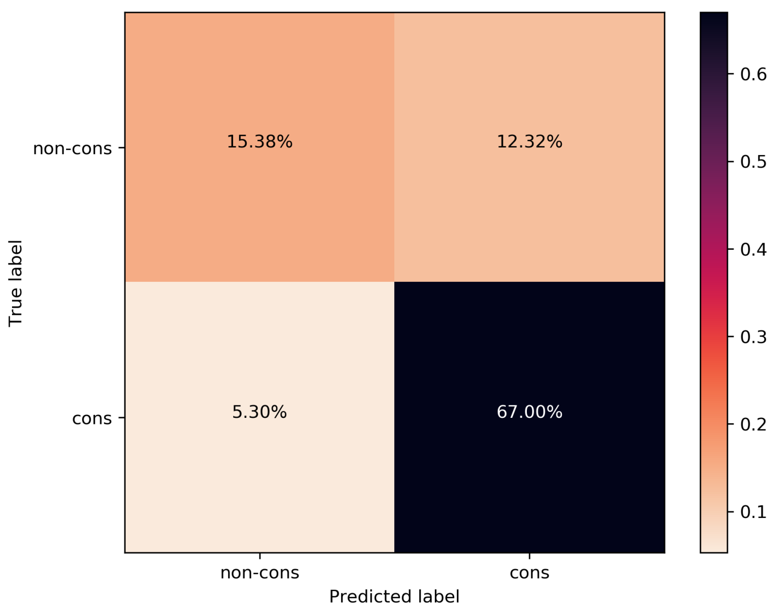


Figure 5.11: Confusion matrix from RASC on the neuron HipSci MAJIQ dataset. 'cons' and 'non-cons' respectively abbreviate constitutive and non-constitutive exons.

the classifier performs well at ruling in that an exon is alternatively spliced, but poorly at ruling that it isn't. This means the classifier is attractive for guiding the investigation of exons, where it wasn't previously known that they are alternatively spliced. This wouldn't be possible with a less specific classifier because further investigation relies on expensive wet-labs experiments for which the number of false positives need to be minimized. When employing RASC in practice, the decision threshold should be chosen such that the specificity is even higher (at the cost of further reduced sensitivity): a false positive rate of 5.3% is still too high for wet lab experiments. Thus, we conclude that our classifier shows attractive properties, but would require further tuning before clinical use.

5.4.4 Cross-condition performance

We evaluate how well the models generalize when trained on a dataset derived from one individual and then tested on a dataset derived[1]

[1]When choosing the samples to evaluate on from the different datasets, care needs to be taken to avoid a subtle data leak: despite the different primary data sources, many of the cassette exons appear as training samples in multipe datasets. This initially lead to the surprising observation that it is better to train on a dataset different from the one the model is evaluated on. We avoid

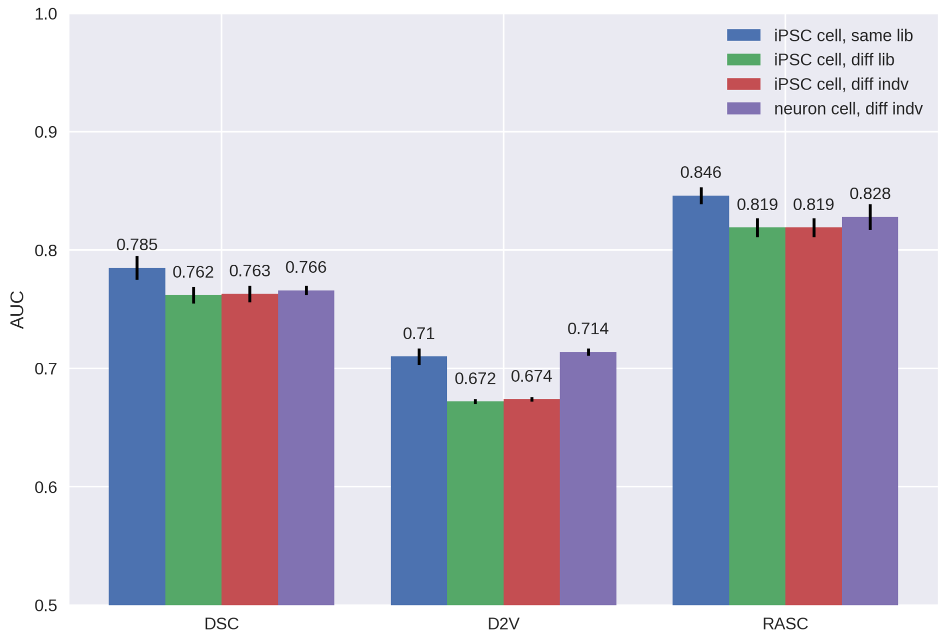


Figure 5.12: Performance when training models on one HipSci dataset and testing on the same dataset as well as three others. Within the bars of one model, going further right means using a dataset which is less similar to the dataset the model was trained on (the left-most bar display performance when 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 indivdiual 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 sensory neuron cells.

- from the same individual, but a different library.
- from a different individual, but the same cell type.
- from a different individual from a different cell type.

We visualize the results in Figure 5.12. Unsurprisingly, performance generally drops when a model is evaluated on a dataset different from the one it was trained on. Surprisingly, this performance drop does not directly correlate with the similarity to the original dataset: the least similar dataset from a different tissue performs best. This is because the baseline performance on the sensory neuron dataset is roughly 2% higher and it experiences only a roughly 1% higher performance drop when training was done on a different dataset. The models generalize extremely well

this issue by efficiently filtering all samples that are in the original training dataset from the three additional datasets via a hash set.

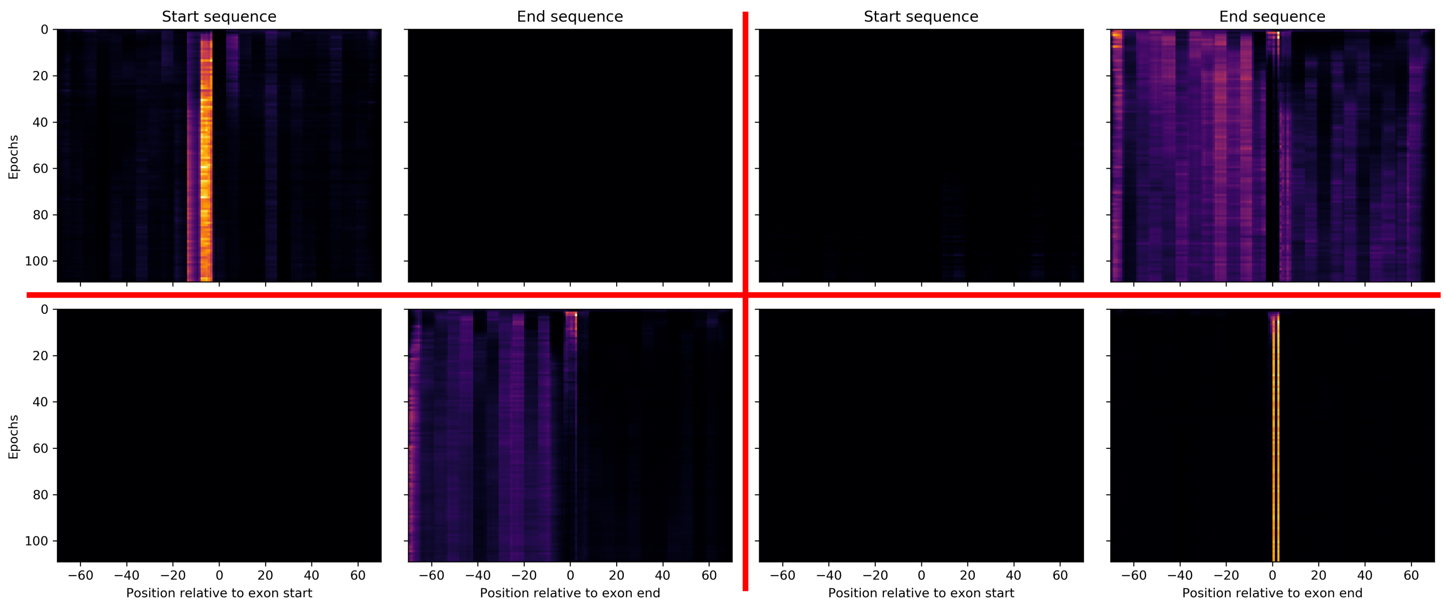


Figure 5.13: Each of the four quadrants shows the distribution of the attention weights for a single attention head. The distribution of attention weights is visualized via one heatmap for the start and one heatmap for end sequence, showing how the distribution of attention weights develops during training. The mean distribution of the attention weights over the test set is displayed.

across tissues and conditions. As previously suggested, this is biologically surprising due to known splicing differences between tissues [**crosstissuesplicing**]. Possibly the datasets don't capture the splicing behaviour which is different across tissues.

However, to more seriously investigate how splicing differs between conditions and tissues, the capability of MAJIQ to quantify differential splicing should be taken advantage of. The best performing model, RASC, could be adapted such that it takes in an exon as well as two different conditions (e.g. via one-hot encoded tissues) and then predicts the change in splicing behaviour between the two conditions.

5.4.5 Interpreting RASC

Heads focus on a single sequence

When using a single attention head, the head needs to split its conceptual 'focus' over the start and end input sequences. This is not the case for multiple attention heads: here single heads may focus on only one sequence as long as there is at

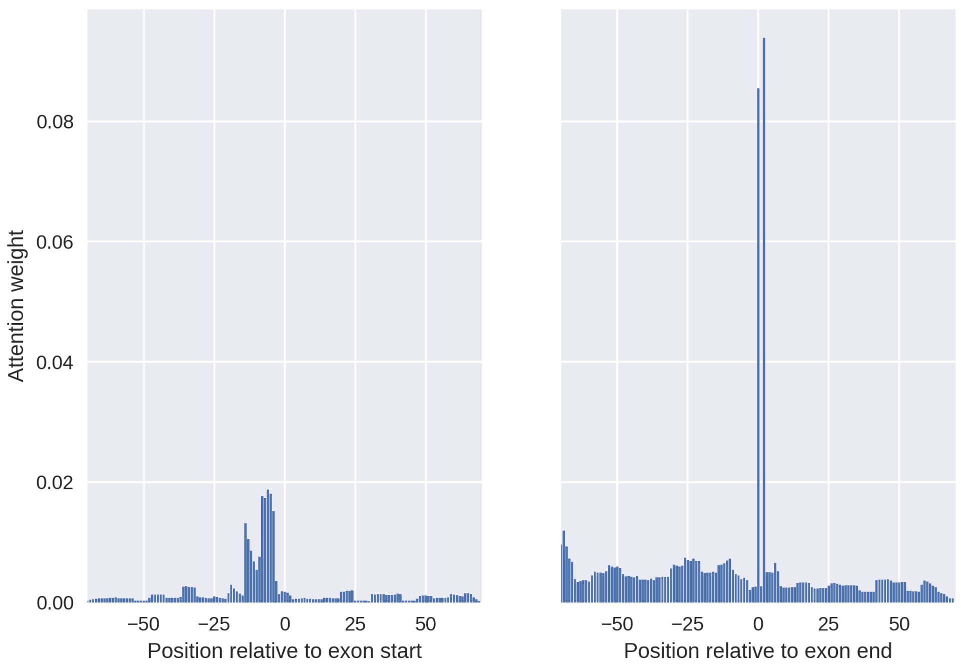


Figure 5.14: Mean attention weights averaged over all test samples and all four attention heads.

least one attention head which also pays attention to the information in the other sequence. We observe exactly this behaviour in Figure 4.15: one of the attention head attends purely to the start sequence and all other attention heads attend purely to the end sequence. Across the 9 cross-validation runs, single attention heads only split their 'focus' 7 out of 36 times. None of the trained models only ever pay attention to only a single sequence. This reassures us that the dataset isn't biased such that the model only needs to take information from one sequence into account.

The sequence which an attention head focusses on generally does not change after the first 5 epochs. Similar observation, that the role of attention heads is decided very early on in training, have also been made in the context of NLP [**sixteenheads**].

What does the model pay attention to?

Averaging the attention weights over all heads at the end of training, we visualized the most attended to parts of the input sequence in 5.14. Clearly, motifs directly

Model	AllPSI	LowPSI	HighPSI
DSC	0.244 $\pm$ 0.036	0.340 $\pm$ 0.039	-77.980 $\pm$ 13.631
D2V	0.120 $\pm$ 0.009	0.155 $\pm$ 0.010	-57.523 $\pm$ 2.771
RASC	**0.486** $\pm$ 0.067	**0.595** $\pm$ 0.074	**-25.470** $\pm$ 5.717

Table 5.4: Mean R^2 for PSI regression task on sensory neuron cell HipSci MAJIQ dataset. $\pm$ denotes standard deviation.

around the exon start and exon end are the most attended to.

The bar chart suggests that more total attention is paid to the end, rather than the start sequence. However, this pattern is reversed for other runs of the models and likely just a sequence of three attention focusing on the end sequence in this particular training run. Additionally, note that the weighting of the outputs of the different attention heads by the unifying matrix W^O is not displayed.

5.4.6 PSI Regression

After having achieved a good level of performance on the classification task, we turn to the regression task. We train the models on the sensory neuron cell HipSci MAJIQ dataset and give the results in Table 5.4.

The results show that the models still fail to explain a lot of the variation in the PSI value: no model is able to explain even half of it. As on the classification tasks, RASC outperforms the other models by a wide margin and all models are significantly worse on explaining the variance for highly included, rather than rarely to moderately included, non-constitutive exons. The latter observation is particularly pronounced: the R^2 score of all models is below 0, showing that they fail to explain the variance even as well as the baseline model whom consistently predicts the mean. Thus, we take these results as indication that the current Splicing Codes can still be significantly improved.

To further compare our model to other work, we repeat the main results from [**d2vsplicing**] in Table 5.5. [**d2vsplicing**] used a dataset derived from mouse sequencing data, originally from [**jha**]. On this dataset, D2V is the second-best performing model, only moderately outperformed by a more finer-grained

Model	Performance
BNN-UDC [**jha**]	0.220
BNN-LMH [**jha**]	0.368
DNN-PSI [**jha**]	0.434
D2V [**d2vsplicing**]	0.594
W2V [**d2vsplicing**]	**0.680**

Table 5.5: Mean R^2 over 5 tissues for PSI regression task on mouse dataset as reported by [**d2vsplicing**].

word2vec model (denoted W2V). Nonetheless, D2V significantly outperforms all other models from the literature it was compared to. Note that the models evaluated by [**d2vsplicing**] comprise all of splicing codes from the last 5 years which we are aware of. However, D2V is significantly outperformed by RASC on our dataset. This indicates that RASC would compare very favorably against all models also evaluated in [**d2vsplicing**].

We also observe that the R^2 values reported by [**d2vsplicing**] are significantly higher: the best performing model W2V is able to explain nearly 70% of the variance. We consider multiple possible explanations for this:

- Splicing behaviour on the mouse dataset is easier to predict because of data processing differences. Like our work, [**d2vsplicing**] use MAJIQ for data processing. While they don't explicitly state the version of MAJIQ they used, MAJIQ has received various updates since their publication. These updates include improvements to the splicing detection algorithm which could affect the outcome of data processing.
- Splicing behaviour on the mouse dataset is easier to predict because of dataset composition differences. The mouse dataset only includes cassette exons, while our dataset also includes constitutive non-cassette exons. Since splicing motifs between cassette and non-cassette exons likely differ, the learning task of our models could be more challenging since they have to learn to recognize both. As a counter point, the mouse dataset only contains 10,000 training samples which might be too few samples to train our models.

- Splicing behaviour on the mouse dataset is easier to predict because of different model inputs. [**d2vsplicing**]'s models D2V and W2V take 300 nucleotides, instead of 70 nucleotides, before and after the cassette exon start and end sites as input. In addition, they are also given analogue 600 nucleotide windows around the exon start and end sites directly flanking the cassette exon. In total their models are given 2400 nucleotides as input, while we give our models a total of 280 nucleotides as input. Widening the input context may help to equalize the performance between the mouse dataset and our dataset.
- Splicing behaviour on the mouse dataset is easier to predict because it is less complex. Increased relative prevalence in exon skipping has been observed for more complex eukaryotic organisms [**splicing_current_perspectives**], giving plausibility to this explanation. However, we observed nearly no cross-tissue performance differences between tissues, when cross-tissue splicing behaviour is also known to vary [**crosstissuesplicing**]. This hypothesis is incredibly challenging to evaluate, as there are many dimensions to splicing and no universal measure for splicing complexity (nor likely ever will be). However, due to the other possible explanations we explored and due to the low cross-tissue performance differences we observed, we evaluate it as unlikely.

While most of these possible explanations are difficult to falsify, increasing the amount of sequence information given to the models is a simple to implement and promising avenue of research which we leave to future work.

Oooooh, the weary traveller draws close to the end of the path.

— Izaro, Emperor of the Eternal Empire

6 Discussion / Conclusion

This work approached the task of predictive alternative splicing behaviour from a Deep Learning perspective. We discussed challenges which should be addressed when estimating PSI, implemented our own method for PSI estimation and constructed splicing quantification datasets based on three different processing methods. We proposed a novel splicing code which introduces the attention mechanism to the field of splicing quantification and reimplemented two baselines models from the literature.

Evaluating the models on a dataset from the literature, we showed that the dataset was systematically biased and that the performance of previous models can be matched by an extremely simple model with two to three order of magnitudes fewer parameters. Moving to the datasets we constructed, we found that only one out of three processing methods provides data of appropriate quality. On this dataset, our newly introduced model outperforms the previous state-of-the-art model by 15%. In further experiments, we analyze the generalization performance of our model to different conditions and discuss its sensitivity and specificity. Finally, we investigate what parts of the most input sequences are most attented to by our model.

Our work further corroborates the importance and difficulty of constructing datasets of appropriate quality to train deep learning models. We also observe that there is a wide variability between the quality of published data processing methods and datasets. The lack of standardized datasets suitable for the training of Machine Learning-based splicing codes was already during the introduction of HEXEvent [**hexevent**]. The most commonly used dataset is based on mouse, and not human data, and not publically made available in an accessive format [**jha**]. As a result, many papers introducing new splicing codes attempt to reconstruct this dataset [**d2vsplicing**], use HEXEvent [**dsc**] or construct their own dataset ad hoc [**cossmo**]. This results in progress being slowed, due to authors each having to construct a new dataset and comparisons not being directly possible or flawed due to implementation differences when reconstructing a dataset.

In contrast, the rapid succession of breakthroughs in Computer Vision and NLP was only possible through the wide use of standardized datasets which allowed a quick iteration of ideas and a fair comparison between them. For all these reasons, we believe that an effort to construct high-quality, standardized datasets for the training of Machine Learning, and particular Deep Learning, splicing codes should be undertaken.

While our newly introduced model is able to improve upon previous models by a wide margin, its prediction accuracy is still perfectible. Incorporating further information sources known to affect splicing like sample tissue or chromatin states [**chromatin**] would likely further improve performance. Even additional hyperparameter tuning may lead to further improvements since our hyperparameter tuning was limited due to computational constraints. Experiments maxing out the batch size (and stabilizing training) or exploring whether a context window larger than 140 nucleotides benefits performance would be particularly interesting.

Towards the end of practical applicability, adapting our model for differential splicing prediction is a promising avenue of future research. Here the already used tool MAJIQ could be exploited to generate a dataset for splicing changes between conditions.

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

7.1 Accession Numbers of HipSci data

The ENA Accession Numbers of the 25 biological replicates belonging to sensory neuron cell lines [**ipscneurons**] 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 [**hipsci**] 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.

7.2 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 [**d2vsplicing**]. We reduced the number of epochs from 20 to 5, as initial tests showed no performance difference between these two values. However, while [**d2vsplicing**] 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 (as the vocabulary only consists of 64 words), we mention them for completeness (as they are also mentioned in [**d2vsplicing**]):

- 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, making it computationally expensive to keep them.
- Negative sampling [**w2v2**] is a technique to reduce the computational cost of backpropagating the gradient updates. 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-dimensional embedding, the matrix representing the hidden weights already

has 3 million weights. This makes backpropagation expensive as all of these weights need to be updated in every step.
When negative sampling is enabled, by default only the weights connected to the word tne network should 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 cost of backpropagating the gradient updates, since only the weights of very few words in the vocabulary are updated.