<table>
<thead>
<tr>
<th>Studieprogram/spesialisering: Informasjonsteknologi, datateknikk</th>
<th>Vårsemesteret, 2008</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forfatter: Øystein Johansen</td>
<td>Åpen / Konfidensiell</td>
</tr>
<tr>
<td>Faglig ansvarlig: Tom Ryen</td>
<td>(signatur forfatter)</td>
</tr>
<tr>
<td>Veileder(e): Tom Ryen, Trygve Eftestøl, Peter Ruoff, Thomas Kjosmoen</td>
<td></td>
</tr>
<tr>
<td>Tittel på masteroppgaven:</td>
<td></td>
</tr>
<tr>
<td>Engelsk tittel:</td>
<td>Gene Splice Site Prediction using Artificial Neural Networks</td>
</tr>
<tr>
<td>Studiepoeng: 30</td>
<td></td>
</tr>
<tr>
<td>Emneord: Genprediksjon, nevrale nettverk, DNA, Arabidopsis Thaliana</td>
<td>Sidetall: ………………..</td>
</tr>
<tr>
<td></td>
<td>+ vedlegg/annet: …………..</td>
</tr>
<tr>
<td></td>
<td>Stavanger, …16. juni 2008 ..</td>
</tr>
<tr>
<td></td>
<td>dato/år</td>
</tr>
</tbody>
</table>
Gene Splice Site Prediction using Artificial Neural Networks

Øystein Johansen

June 16, 2008
Abstract

Gene prediction is the process of finding the location of genes and other meaningful subsequences in DNA sequences. This process is time consuming and expensive when done by biochemical methods and genetics. An other approach to the process of predicting genes is ab initio gene prediction. With this approach genes are predicted by analysing the sequence of nucleotides in the DNA using a statistical method. The process can then be carried out in a computer system, which is faster and less expensive. A computational tool to predict genes in DNA sequence is therefore of high importance and great value to the biologists.

The DNA molecule contains subsequences that codes protein chains. The proteins form the functionality of the organism. The subsequences that code to these proteins are called genes. In eukaryotic cells, the gene sequences consist of exons and introns. The exon part is coding to proteins while the intron part is rejected in the splicing process. The transition between an exon to an intron sequence is a splice site. The proposed system in this thesis, will try to predict the splice sites in the genes. The gene sequences used in this thesis are from the model plant Arabidopsis thaliana.

Artificial neural network is a mathematical method known from artificial intelligence and pattern recognition. It is a method that can be used as a general function approximator. In a training phase, the neural network is presented input patterns and corresponding desired outputs, and these are used to adjust weights inside the neural network. Later, the neural network can approximate an output from an arbitrary input pattern. Artificial neural networks have shown to be usable in many application, and it has also been used in gene prediction.

The proposed system connects an artificial neural network to a gene sequence, and the system tries to predict the splice sites in the sequence. A window of 60 nucleotides slides over the gene, and the neural network will evaluate the pattern in this window to predict if there are any splices sites in this pattern. The output of this evaluation is a numeric value, and the values are accumulated for each nucleotide as the window slides over the gene. The accumulated score is used as an indicator of where the splice sites are located. To find the exact location of the splice site, a second order polynomial function is fitted through successive data points in the splice site indicator. The top location of this parabola is used as the predicted location of the splice site.
The system has been developed and some experiments have been performed. The system has been trained on a data set of 15551 genes, and a performance benchmark of the neural network is done at a distinct set of 5000 genes. The best neural network achieves a sensitivity of 0.851, a specificity of 0.844, and a correlation coefficient of 0.568. These are reasonable measurements considering that no prior knowledge about special splice site signals were given to the system.
Chapter 1

Introduction

In this study we try to utilize the technology of artificial neural networks in gene prediction. An artificial neural network is trained and then used to find splice sites in genes of the model plant Arabidopsis thaliana.

1.1 The issue

Gene prediction has become more and more important as more organisms are becoming sequenced. DNA sequences submitted to databases are often already characterized and mapped when they are submitted. A molecular biologist already has used genetics and biochemical methods and computational methods to find genes, promoters, exons and other meaningful subsequences in the submitted material. However, the number of sequencing projects is increasing, and a lot of DNA sequences have not been mapped or characterized. Having a computational tool to predict genes is therefore of great value, and can save the biologist for a lot of expensive and time consuming experiments.

1.2 Gene prediction

There exist several methods to predict where in a DNA sequence the genes can be located. There are basically three fundamentally different methods to predict these genes.

Extrinsic gene prediction. This method searches for known sequences that will form explicit proteins. This is also called evidence based methods. The process is often carried out by reverse translating the protein and
then comparing these to the DNA sequence. Where the reverse translated protein sequence matches the DNA to be analysed, the matching subsequence is marked as a candidate coding region. Each candidate coding region is then further analysed, together with other information, to predict the location of the gene. BLAST\cite{1} is a computational tool for performing these kinds of analysis.

**Comparative genomic gene prediction.** These methods are based on comparing two species to find common subsequences in their DNA. Because protein coding sequences are more resistant to mutations, genes can be detected by comparing the genome of the two related species.

**Ab initio gene prediction.** A method analysing the sequences of bases in the DNA using only a computational model without any comparison to existing data. The method uses probabilistic methods such as discrete Fourier transform (DFT), support vector machine (SVM) and hidden Markov models. The method described in this thesis is also an *ab initio* method of gene prediction.

### 1.3 Artificial neural networks

An artificial neural network (ANN), or simply neural network (NN), is a computational model of how a biological neural network works. To model the biological neural network a set of neurons are defined as numerical values. A set of numeric weights represents the connections between the neurons. These weights can be adjusted by a training method, such that the neural network can learn simple concepts.

Artificial neural networks has been used in several real life applications like image data processing, traffic control, regression analysis, pattern recognition and classification like hand written character recognition.

Some studies have also tried to utilize neural networks in gene prediction. According to Reese \cite{18} the first attempt was done by Brunak et al.\cite{8}. Later, similar methods have been in other works (References \cite{18,14,12,6}).

### 1.4 Scope of work

The work described in this thesis tries to utilize an artificial neural network to predict where the splice sites of a gene can be located. It is a full *ab initio* method, and does not use any prior information about splice site signals such as 'GT' or 'GC' for the donor splice sites, or 'AG' for the acceptor splice sites.
1.5 Outline

Chapter 2 will describe the background for this study. The first section of this chapter gives a brief introduction to the microbiology of cells, how chromosomes store the genetic information and how the cell transcribes and translates the genetic information to proteins. The last section of this chapter briefly explains how an artificial neural network can be trained and used to make general approximations. Chapter 3 describes the proposed method of utilizing an artificial neural network to predict the splice sites in a gene. The results of the training are presented in chapter 4. An outlook and a conclusion are given in the last chapter.
Chapter 2

Background

This chapter will describe the biological concepts that make the foundation of bioinformatics and gene prediction. A brief description of artificial neural networks and how their weights are adjusted to train their abilities to learn concepts is given at the end of this chapter.

First part of this chapter briefly describes the biology of cells, with focus on the process of how a DNA codes to the proteins, a process often called the central dogma of molecular biology. It is the three step process of generating proteins from the DNA molecule. This part will briefly explain the important terms to understand this process.

The second part of this chapter describes the term gene prediction, what it is and why it is important to us. The last part gives a brief introduction to artificial neural networks, what neural networks can do and briefly how they work.

2.1 Biological background

2.1.1 Cells

All organisms on this planet consist of cells. There are two types of cell structures, prokaryote cells and eukaryote cells. The prokaryote cell has not developed any cell nucleus, and this cell structure only forms simple life form like bacteria and some other microorganisms. In this study we only consider the eukryotic cell structure, which has developed a cell nucleus. Inside the cell nucleus are the chromosomes, and these chromosomes contains the DNA sequence of the organism. Figure 2.1 shows a typical plant cell.
2.1.2 Amino acids and proteins

Proteins are large organic compounds built up by amino acids. Amino acids are organic compounds that consist of an amino group, a carboxyl group and an organic side chain. There exist more than 100 natural amino acids, however only 20 of these form the fundamental building blocks of proteins [2]. These 20 fundamental amino acids are the same for all living organisms, and are called $\alpha$-amino acids. Proteins are long chains of different amino acids connected to each other through a covalent bond.

Proteins in the cells make most of the functions in the organism, like mechanical functionality, immune defence system functionality, catalytic functions (enzymes) and cell controlling functions. Many of the functions provided by the proteins are crucial for the organism. Proteins can also interact with each other to achieve a particular functionality.

The cells of the organism build their own proteins, but some organisms require to absorb essential amino acids from the outside. For those organisms, such as humans, these essential amino acids come from food.
2.1.3 DNA

DNA, *Deoxyribonucleic acid*, is a huge molecule inside the nucleus of eukaryote cells. It is the most important chemical part of chromosomes. The DNA molecule contains all the genetic information for the organism. The DNA molecule and its genetic functionality, was first proved by Oswald Avery in 1944 [2].

The molecule consists of a sugar-phosphate backbone structure. The backbone structure has two strands which forms a double helix. To the strands there are connected four different base units, nucleotides, either *adenine* (A), *thymine* (T), *guanine* (G) or *cytosine* (C).

In the strands in the double helix the bases are coupled to each other with a hydrogen bond. A T base is always paired with an A base and vice versa. Also, a G base is always paired with a C base. Each A and T pair is coupled by a double hydrogen bond, while C and G pairs are connected with a triple hydrogen bond. See Figure 2.2.

![Double helix of DNA](image)

Figure 2.2: DNA double helix and the base pairs. (Figure from *Molecular Biology of the Cell, 5th Ed.* [5])
Sugar backbone

The backbone structure of the DNA molecule is made of phosphate group and sugar. The sugar is carbohydrates and each carbon on the sugar of a nucleotide is numbered. The number is followed by a prime mark. The sugar structure forms a pentagon of bindings, and is therefore called pentose or 5-prime carbon. The DNA backbone structure will therefore have a direction based on the orientation of the sugar structure. The sugar of the backbone is linked together through the 5’ carbon and the 3’ carbon. The backbone will therefore have a 3’ end and a 5’ end. The DNA double helix has two backbones, or strands. These two strands has opposite directions.

Chromosomes

The DNA molecule holds the genetic information of the organism. The DNA molecules are very long, and they are winded into threadlike packages called chromosomes. The DNA double helix is spooled around histones, a simple protein structure, forming a chromatine. This chromatine string is winded to nucleosomes which again forms a helix of condensed chromosome. This condensed helix is then packed into a chromosome. See Figure 2.3 or one of the references [4] or [3]. Different species has different number of chromosomes. The chromosomes reside inside the nucleus of the cell.

RNA and DNA

There are two kinds of pentoses which are used as a molecule backbone, β-D-2-deoxyribose and β-D-ribose. The β-D-2-deoxyribose forms a DNA molecule and the β-D-ribose forms a ribonucleic acid molecule or RNA molecule. The RNA molecules have a one strand backbone structure, and contain the same four nucleotide bases as the DNA, except that thymine (T) is replaced with a similar base called uracil (U).

2.1.4 Genes

The gene is the subsequence of the DNA molecule, with its promoter[1] that code for functional behaviour. This means that the gene codes for amino acids that generate proteins. There are also some genes that code for tRNA and ribosome-RNA. The glossary in Molecular Biology of the Cell 4th ed. [4] states:

[1] A promoter is a regulatory nucleotide sequence upstream the coding region.
Figure 2.3: A typical DNA double helix and how it is spooled and wound to form a chromosome. (Figure by National Institutes of Health, public domain)
**Gene** Region of the DNA that controls a discrete hereditary characteristic, usually corresponding to a single protein or RNA. This definition includes the entire functional unit, encompassing coding DNA sequences, noncoding regulatory DNA sequences and introns.

This definition does in no way exclude the promoter from the gene, but rather states that the promoter region is a part of the gene. However, the definition is changed in the 5th edition of the same book [5]. In the fifth edition the same glossary says:

**Gene** Region of DNA that is transcribed as a single unit and carries information for a discrete hereditary characteristic, usually corresponding to (1) a single protein (or a set of related proteins generated by variant post-transcriptional processing), or (2) a single RNA (or set of closely related RNAs).

Promoter is an upstream subsequence that has a controlling functionality. By the latter definition, the gene consists only of the transcribed part. The promoter is never transcribed, so the gene only consists of the exons and introns by this last definition. In this report we will treat the promoter as a part of the gene. The exon part is usually coding to the amino acids and thereby to the proteins, while the intron parts are removed from the RNA in the splicing process.

A simplified figure of the gene sequence in the DNA string, can be seen in Figure 2.4.

### 2.1.5 From DNA to proteins

The process of forming a protein from a DNA sequence consists of three steps; the transcription, splicing and translation. The step by step process is shown in Figure 2.5, and described in details below. This stepwise process of forming proteins from DNA is often called the central dogma of molecular biology.

**Transcription**

An enzyme in the cell nucleus, called RNA-polymerase II, makes a complementary copy of a DNA region. This copy is called pre-mRNA. To initiate this process, about 50 transcription factors and co-activators bind to promoter sequences of the gene. These factors are joined by general transcription
Figure 2.4: A typical coding region in a segment of eukaryotic DNA. The chromosome, the DNA double helix, the gene and the exon and introns. (Figure by National Institutes of Health, public domain)

Figure 2.5: The process of coding a DNA sequence into a protein. (Figure by Rüegg and Hurlimann [19]. Modified and used with permission)
factors, SRBs and RNA polymerase II at the promoter to assemble a transcription initiation complex. Together they physically open the DNA double helix and read the bases of the DNA sequence. The two strands are denoted as coding strand and template strand in this process. The RNA-Polymerase II moves in 3’ to 5’ direction of the template strand. The nucleotides are copied to the mRNA as the complement of the template strand nucleotides, such that the resulting mRNA becomes a copy of the corresponding coding strand.

Transcription begins at transcription start site and ends at transcription stop site. The process makes an exact copy, except that the thymine (T) base is copied to an uracil (U) base.

The transcription process with the initialisation, elongation and termination, is shown in Figure 2.6.

**Splicing**

The transcription makes a RNA copy of exons and introns. The exons are usually the protein coding part, and all the introns are cut off from the pre-mRNA to form a mRNA (messenger-RNA). This process is called splicing, since all exons are spliced together. The RNA splicing is performed by the spliceosome. Spliceosome is a large assembly of proteins and RNA molecules. The transition between exon and intron, and the transitions between intron and exon are called donor splice site and acceptor splice site, respectively.

There is little general resemblance between the sequences of the different introns, but the splice sites have some short sequences that make a biochemical reaction with the spliceosome that forms the splice site. These sequences are found close to ends of the introns. The splicing step in the process must be considered of great importance for this study. The neural network will be trained to recognize patterns of nucleotides around the slice sites. More information about the splicing operation can be found in *The Molecular Biology of the Cell, 5th Ed.* [5].

**Translation**

The final step to generate protein is the translation. This translation to proteins is done by the ribosome in the cytoplasm. The ribosome reads triplets of nucleotides, codons, from the mRNA, which forms amino acids to build proteins.

After the introns have been removed during the splicing, a protein complex cleaves the mRNA, and adds a poly-A tail to the 3’ end. A poly-A binding protein and a cap binding complex, protects the mRNA and aids the
Figure 2.6: A simplified diagram of the transcription process. The progress starts with a compound of transcription factors that connects to the enhancer area of the DNA, and the RNA Polymerase II (denoted RNAP in the figure) attaches at the promoter (a). The elongation step (b) reads the DNA template strand and makes an RNA copy of the gene. The process is terminated when the RNA Polymerase II reaches the transcription stop site. The RNA polymerase II is released from the DNA molecule. (c). (Wikimedia Commons, public domain)
mRNA to the cytoplasm of the cell. There, an initiation factor binds to the cap while other proteins bind to the mRNA and the poly-A tail. The mRNA is now prepared for translation. Initiation factors, the 40S ribosomal subunit, and the initiator tRNA scan for the nearest ATG start codon. Translation begins when the 60S subunit binds to form a ribosome, and ends when the ribosome encounters a stop codon. Table C.1 show how different codons are coded to amino acids.

2.1.6 Arabidopsis thaliana

*Arabidopsis thaliana* is a small flowering plant. It has a relatively short life cycle, and it is therefore often used as a model organism in molecular biology and genetics. The genome\(^2\) of the plant is also one of the smallest among other plants. It has only five chromosomes\(^3\). Its genome is fully sequenced.

2.2 Gene prediction

Among many different methods of detecting meaningful signal in the DNA sequence in a uncharacterised DNA sequence, *gene prediction* is one of the most important. Gene prediction is the process of finding the most probable set of exons and introns, or other meanings, in a given DNA sequence.

Sequences deposited to databases are usually already characterized. This means that someone has already studied the sequence and found a meaning of the sequence, using molecular biology, genetic, or biochemical methods. As more projects of sequencing all kinds of organisms, there will be a lot of DNA sequences that is not characterized.

Computational tools for predicting exon, introns, genes, open reading frames, splice sites and promoters helps molecular biologists find the meaning in uncharacterised DNA sequences. Computational tools for gene prediction will therefore become more and more important as more organisms are sequenced.

The proposed method in this report is an *ab initio* gene prediction method; a method based on the nucleotide sequence and is not expressed on the protein or amino acid sequences. The *ab initio* methods are carried out using computational tools.

---

\(^2\)The *genome* is the total genetic material of an organism. It is a term constructed by the words *gene* and *chromosome*.

\(^3\)There also resides some genes in the mitochondrion (see Figure 2.1) and plastid of the cell.
2.2.1 Generalized Hidden Markov Models (GHMM)

One of the most popular and successful methods in gene prediction is the Generalized Hidden Markov Model [23]. The hidden Markov model is a statistical model. The modelled systems are assumed to be a Markov process, which means that the state of the system is randomly evolving with probabilities depending only of the previous states. The generalized hidden Markov model takes also into account other dependencies, like the number of states step since the last state transition.

This statistical model is utilized on DNA sequences, where exon, intron, intergenetic region and promoter are used as different states in the model. Such models have shown good results for gene prediction and have been popular in gene prediction.

At our university, the method has been studied by Rüegg and Hürlimann [19], Beck and Frei [7], and by Strässle and Boos [22].

2.2.2 Neural networks

Another method used in gene prediction is artificial neural networks. In neural networks a structured set of numerical weights get adjusted based on a pattern or feature vector where the desired output can be obtained. This is known as the learning or training of a neural network. Later, the trained neural network can be used to estimate an output based on the weights adjusted in the training.

Utilizing artificial neural networks to predict genes is nothing new, and several methods have been tried. The different genes of an organism are quite different to each other, so it is hard to make a general system which can recognize a gene sequence. In the same way there is not much in common of the different exons and introns. They also differ in length, and the length of an exon can be from a few nucleotides to more than 1000 nucleotides. A system to recognize exons sequences is therefore also hard to develop. A feasible method is to make a neural network predict the splice sites and the translation start and termination sites. Predicting the location of the splice sites will also predict the locations exons and introns.

The first documented attempt to utilize neural network to predict splice sites in DNA sequences was performed by Brunak et al. [8]. In this study artificial neural networks were used to predict splice sites in human genes. A later study by Hatzigeogiou et al. [12] used a similar system of predicting functional sites using neural networks. It used a sliding window over human genome sequences. Reese et al. [18] studied a system, Genie, which used a generalized hidden Markov model to predict the gene locations and then
used a neural network to improve the splice site predictions. The splice site predictor for *Arabidopsis thaliana* using neural networks has also been developed by Hebsgaard et al. [14].

In this thesis we have tried to train an artificial neural network to recognize splice sites in gene sequences of DNA strings. Artificial neural networks are therefore described further in the next section.

### 2.3 Artificial Neural Networks

Artificial neural networks (ANN), or just neural networks is a technology utilized in the development of artificial intelligence. The neural network technology is inspired by how the brains of humans and animals work. The brain is composed of millions of neurons, and these neurons are connected to each other by axons and dendrites. The connections are adaptive, which means that the connection structure is dynamically changing. Changes of the connections is what we call learning.

An artificial neural network is similar, where the strength of couplings between artificial neurons are described by numerical weights. One of the most used structures is the multilayer neural network. The neurons are usually modelled in three layers, where the first layer is called the input layer, the intermediate layer is called the hidden layer and the last layer is called the output layer. See Figure [2.7] The neurons in this model are sometimes called units or nodes. The nodes of each layer are connected to each other with adjustable weights. The process of adjusting these weights is called training. The numeric value of a unit is the sum of all its input connections times the weight of that input connection. The numeric value of each neuron is then adjusted with a non-linear activation function.

An artificial neural network can be trained to recognize input patterns, give the desired result, by adjusting the weights. A neural network is therefore a general approximation function. A neural network approximation function can have an arbitrary number of inputs and outputs.

#### 2.3.1 Feedforward calculation

After the weights have been trained in the neural network, it can be used to evaluate an input feature vector and predict an associated outcome vector. The process of taking an input vector and predicting an output is called the feedforward operation. The feedforward operation is given by Equations 2.1, 2.2, 2.3 and 2.4. The input vector is denoted by $\mathbf{x}$. A vector $\mathbf{net}_1$ is a temporary array called the activation and the indices 1 and 2 indicate the
Figure 2.7: Structure of a multilayer neural network. This network has three input units, four hidden units and three output units. The units, or neurons are connected through adjustable weights. Adjusting the weights is considered as training of the neural network.
activation for the hidden layer and the output layer, respectively. The vector \( y \) is the resulting vector from the hidden layer. The weight matrices are named \( W_1 \) and \( W_2 \) for the hidden and output layer, respectively. The bias vectors are named similarly, with \( b_1 \) and \( b_2 \). The activation function, \( f(x) \), is a non-linear function.

\[
\text{net}_1 = W_1 x + b_1 \tag{2.1}
\]

\[
y = f(\text{net}_1) \tag{2.2}
\]

\[
\text{net}_2 = W_2 y + b_2 \tag{2.3}
\]

\[
z = f(\text{net}_2) \tag{2.4}
\]

These equations assume that there are three layers in the neural network. If there are more than three layers, these equations repeat. The algorithm for the evaluation operation is given as pseudo code in Appendix A.

This study will take a subsequence of the DNA, calculate a feature vector, and predict score for whenever or not there is a splice site in this subsequence.

### 2.3.2 Backpropagation

The backpropagation algorithm is the algorithm that adjusts the weights in the neural network. The weights are adjusted with a gradient descent algorithm, where the training error, \( J \), is defined by Equation 2.5, where \( z \) is the output of the feedforward operation and \( t \) is the desired output vector. The training error is a function of all adjustable weights, \( w \).

\[
J(w) = \frac{1}{2}\|t - z\|^2 \tag{2.5}
\]

The weights are updated by the rule given by Equation 2.6 where \( \Delta W \) is the change in weights and \( \eta \) is the learning rate parameter.

\[
\Delta W = -\eta \frac{\partial J}{\partial W} \tag{2.6}
\]

The equation can be expanded by the chain rule, which makes the calculation trivial if the non-linear activation function is selected carefully. For further details about the backpropagation algorithm, see Duda, Hart, Stork [11], Haykin [13] or Kartalopoulos [17]. There is also a more detailed algorithm listed as pseudo code in Appendix A.2.
Chapter 3

System setup and algorithms

The first step in this study is to build a set of computational tools that can train neural networks to recognize splice sites, and then investigate DNA strings for potential splice sites using the same neural network. This chapter describes the details of the neural network setup, its training and usage.

The first part of this chapter describes the terminology used in this report. After this, our proposed method is explained in details. The neural network will be described in details, with input layer, hidden layer, and output layer, the input feature extraction, activation functions, and backpropagation. This part will describe how the neural network is connected to genes data structure by a sliding a window over the gene data structure. The gathering of data for training and benchmarking is described. Algorithms for the training and evaluation of genes are given as pseudo code.

3.1 Terminology

To avoid some potential misunderstanding, some important terms are defined first in this chapter.

3.1.1 The definition of the term gene

The gene, as we know it from biology, was defined in section 2.1.4. However, the data needed for the different operations has to be stored in a data structure. This data structure is also called a gene, and only stores the data necessary to predict splice sites. In this chapter, the term gene refers to the data structure gene, and not necessarily the biological gene. There are some small differences of a biological gene and a gene represented in the developed tools. These differences are described in Figure 3.1.
Figure 3.1: The typical biological gene (a). According to the definition in section 2.1.4, the gene contains the promoter, the exons, and the introns. The part from translation start site to transition stop site is called the open reading frame. The data structure of the gene (b) is much simpler. The stored sequence is only the exons and introns. The initial exon and terminal exon are treated like the other exons. In the data structure, no information about the translation start site or translation stop site is stored.
3.1.2 Operations

**Forward calculation.** This is the feedforward calculation of the neural network. It is the operation done by the neural network on a window of 60 nucleotides. This operation returns a vector of two values, a predicted score for donor splice site and acceptor splice site, respectively. This operation calculates inputs from the window, and calculates the output through the neural network as in Equations 2.1, 2.2, 2.3 and 2.4, and Algorithm 5.

**Evaluation.** This operation is looping a sliding window over all positions in one gene, and then giving a score contribution to a running total for every nucleotide in the gene. This operation is therefore done by the `EVALUATEGENE()` function on a gene. An evaluation is considered to be both finding the *splice site indicators* and based on these indicators find the splice sites.

**Benchmarking.** A benchmark is the operation of doing an evaluation of all genes in a data set. The data set can be any collection of genes, but one special data set for this purpose has been collected. This special set is called the *benchmark data set* and the genes in this data set are not used in the training operation. This operation returns benchmark number or performance measures like *sensitivity*, *specificity* and *correlation coefficient*. (See section 3.6.2 for more details on these measures.) Benchmarking is therefore an operation done with a neural network and a data set, and is done to measure the performance of the neural network.

**Training.** This is the operation of training a neural network. The term *training* is used either when training is done on a single gene or a whole data set of genes. Training calls `BACKPROPAGATION()` repetitively.

3.2 Training data and benchmarking data

Based on data from The Arabidopsis Information Resource (TAIR) release 7 website [15], we compiled a certain set of genes. TAIR is an on-line database resource of genetic and molecular biology data of the model plant *Arabidopsis thaliana*. 
3.2.1 Excluded genes

All genes that contain unknown nucleotides were excluded from the data set. In addition, all single exon genes were excluded. Further, all genes with very short exons or introns were excluded. By "short" we mean 30 nucleotides or less. These genes were excluded to avoid very short exons or very short introns. Excluding these genes also simplifies the calculation of desired outputs, because it then can not occur more than two splice sites in a window.

For genes with alternative splicing, only one splicing variation was kept. The only kept splice variation was the one with the highest numeral value. For instance, AT5G58140.4 was included in the data set, but AT5G58140.1, AT5G58140.2 and AT5G58140.3 were excluded.

A further description of all excluded genes can be found in Appendix B.

3.2.2 Training data set and benchmark data set

The remaining data set, after exclusion of some genes, consists of 20551 genes. This set is divided into a training data set and a benchmarking data set. The training set and the benchmark set have 15551 and 5000 genes, respectively. The number of genes in each set is chosen such that the benchmark set is large enough to achieve a reliable performance measure of the neural network. This splitting was done at random. Both data sets contain genes from all five chromosomes.

3.3 Training method

The neural network training is done using standard backpropagation. This section describes how the neural network inputs were calculated and how the desired output was obtained.

3.3.1 Sliding window

For each gene in the training set, we let a window slide over the nucleotides. The window moves one nucleotide each step, covering a total of $L_G - L_W + 1$ steps, where $L_G$ is the length of the gene and $L_W$ is the length of the window. As mentioned earlier, the length of the window is 60 nucleotides in this study.
3.4 Neural network

The main premise in this study is to use a window of nucleotides that moves stepwise over the sequence to be analysed. The inputs to the neural network are calculated from the input calculator. The input calculator extracts the features of the sliding window, as described in section 3.4.4. For each step of the sliding window the neural network will give an output score if it recognizes there is a splice site in the window. A diagram of the entire prediction system is shown in Figure 3.2. The window size is chosen to be 60 nucleotides. This is hopefully wide enough to find significant patterns on both sides of the splice site. A bigger window will make the neural network bigger and thereby harder to train. Smaller window would maybe exclude important information around the splice site.
3.4.1 Network topology

The neural network structure is a standard three layer feedforward neural network. There are two output units corresponding to the donor and acceptor splice sites, 128 hidden layer units and 240 input units. The 240 input units were used since the orthogonal input scheme uses four inputs for each nucleotide in the window. The number of hidden layer units were chosen based on a previous work by Johansen [16], that showed that neural network with this size were able to learn complex concepts. Also, the neural network program code was reused from that study, and the code was optimized for this number of hidden units. There is also a bias signal added to the hidden layer and the output layer. The total number of adjustable weights is therefore \((240 \times 128) + (128 \times 2) + 128 + 2 = 31106\).

3.4.2 Activation function

The activation function is a standard sigmoid function, shown in Equation 3.1. The \(\beta\) values for the sigmoid functions are 0.1 for both the hidden layer activation and the output layer activation. Preliminary experiments were performed to test the effect of these values. These tests indicated that 0.1 was a suitable value. Previous work by Johansen [16] has shown that the effects of the \(\beta\) values are negligible with respect to how fast the weights converge.

\[
f(x) = \frac{1}{1 + e^{-\beta x}}
\]  

(3.1)

When doing forward calculations and backpropagation, the sigmoid function is called repetitively. It is therefore important that this function has a high computational performance. A fast and effective evaluation of the sigmoid function can improve the overall performance considerably. To improve the performance of the sigmoid function, a precalculated table for the exponential function is used. This table lookup method is known to be very fast, and with an acceptable accuracy.

3.4.3 Backpropagation

The neural network was trained using a normal backpropagation method as described in Duda, Hart, Stork [11], Haykin [13] or Kartalopoulos [17]. There is no momentum used in the training. We have not implemented any second order methods to help the convergence of the weights.

---

1This kind of neural network has several names, such as multilayer perceptrons (MLP), feed forward neural network, and backpropagation neural network.
3.4.4 Input to the neural network

For each nucleotide in the sliding window, we have four inputs to the neural network. The four inputs are represented as an orthogonal binary vector. (A=1000, T=0100, G=0010, C=0001). This input description has been used in several other studies [12, 14], and is described in Baldi and Brunak [6]. This input system is called orthogonal input, due to the orthogonal binary vectors. According to Baldi and Brunak [6] this is the most used input representation for neural networks in the application of gene prediction. This input scheme also has the advantage that each nucleotide input is separated from each other, such that no arithmetic correlation between the monomers needs to be constructed.

3.4.5 Desired output and scoring function

The task is to predict splice sites, thus the desired output is 1.0 when a splice site is in the middle of the sliding window. There are two outputs from the neural network; one for indicating acceptor splice site and one for indicating donor splice site.

However, if it is only a 1.0 output when a splice site is in the middle of the window, and 0.0 when a splice site is not in the middle of the window there will probably be too many 0.0 training samples that the neural network would learn to predict everything as 'no splice site'. This is why we introduce a score function which calculates a target output not only when the splice site is in the middle of the window, but whenever there is a splice site somewhere in the window. We use a weighting function where the weight of a splice site depends on the distance from the respective nucleotide to the nucleotide at the window mid-point. The further from the mid point of the window this splice site is, the lower value we get in the target values. The target values decrease linearly from the mid point of the window. This gives the score function as shown in Equation 3.2

\[ f(n) = 1 - |1 - \frac{2n}{L_W}| \quad (3.2) \]

If a splice site is exactly at the mid point, the target output is 1.0. An example window is shown in Figure 3.3.

In some cases there may be two splice sites in a window. It is then one acceptor splice site and one donor splice site.
Figure 3.3: Score function to calculate the desired output of a sliding window. The example window in the figure has a splice site after 21 nucleotides. This makes the desired output for the acceptor splice site 0.7.

3.4.6 Algorithm for training on one single gene

The algorithm for training on a single gene is very simple. The program loops through all the possible window positions. For each position, the program computes the desired output for the current window, computes the neural network input and then calls Backpropagation() with these computed data. See Algorithm 1.

Algorithm 1: Training the neural net on one single gene

```
1: procedure TrainGene(NN, G, η) ▷ Train the network on gene G
2: n ← length[G] − LW
3: for i ← 0 to n do
4:   W ← G[i..(i + LW)] ▷ Slice the gene sequence
5:   desired ← CalculateDesired(W)
6:   input ← CalculateInput(W)
7:   Backpropagation(NN, input, desired, η)
8: end for
9: end procedure
```

In this algorithm, NN is a composite data structure that holds the neural network data to be trained. G is one specific gene that the neural network should be trained on. The first integer variable n is simply the total number of positions of the sliding window. LW is the number of nucleotides in the sliding window, which in this study, is set to 60 nucleotides.

The desired output is calculated as described in Section 3.4.5 and is listed
Algorithm 2 Calculating the desired output based on a given window

1: function \text{CalculateDesired}(W) \triangleright \text{Calculate desired output}
2: \hspace{1em} D \leftarrow [0.0, 0.0] \triangleright \text{Initialize the return array}
3: \hspace{1em} prev \leftarrow \text{IsExon}(W[0]) \triangleright \text{Boolean value}
4: \hspace{1em} \text{if} prev \text{ then}
5: \hspace{2em} j \leftarrow 0
6: \hspace{1em} \text{else}
7: \hspace{2em} j \leftarrow 1
8: \hspace{1em} \text{end if}
9: \hspace{1em} \text{for} \ i \leftarrow 1 \ \text{to} \ L_W \ \text{do}
10: \hspace{2em} this \leftarrow \text{IsExon}(W[i])
11: \hspace{2em} \text{if} \ prev \neq this \text{ then}
12: \hspace{3em} D[j] \leftarrow D[j] + 1 - |1 - \frac{i}{60}| \triangleright \text{Score function}
13: \hspace{3em} j \leftarrow 1 - j \triangleright \text{Flip the index 0 to 1, 1 to 0}
14: \hspace{2em} \text{end if}
15: \hspace{2em} prev \leftarrow this
16: \hspace{1em} \text{end for}
17: \hspace{1em} \text{return} \ D
18: \text{end function}

3.5 Evaluation method

The evaluation of a gene is simply the forward calculation performed for all the window positions in that gene. The neural network outputs are accumulated as an indicator value for each nucleotide in the gene.

3.5.1 Sliding window

Because the neural network is trained to recognize splice sites in a 60 nucleotides wide window, the forward calculation process is also performed on the same sized window. The window slides over the gene in the same way as in the training procedure.
3.5.2 Cumulative output and normalization

The sliding window moves over the gene and forward calculates whenever there is a splice site in the window. A nucleotide gets a score contribution from 60 outputs corresponding to the sliding window passing over it. All these outputs are accumulated.

The accumulated output is then normalized. Most of the nucleotides will get a contribution from 60 different window positions, and these nucleotides are normalized by dividing the cumulative output by the area under the score function (30.0). These normalized cumulative scores are called acceptor splice site indicator and donor splice site indicator.

3.5.3 Algorithm for evaluating one single gene

The pseudo code of the evaluation of a gene is given in Algorithm 3. The algorithm contains two loops. The first loop a slides a window over all positions in the gene and adds up all the predictions from the neural network. The second loop normalizes the splice site indicators.

Algorithm 3 Evaluation of gene

```plaintext
function EVALUATEGENE(NN, G) ▷ Calculate splice site indicators
    in: Neural network (NN), Gene (G)
    out: Two arrays, D and A, which contains the donor and acceptor splice site indicator.

    n ← length[G] − LW
    for i ← 0 to n do
        W ← G[i..(i + LW)] ▷ Slice the gene sequence
        input ← CALCULATEINPUT(W)
        pred ← EVALUATE(NN, input) ▷ Gets predicted output
        for j ← i to LW + i do
            D[j] ← D[j] + pred[0]
        end for
    end for
    for i ← 0 to length[G] − 1 do ▷ Normalizing loop
        D[i] ← 2D[i]/LW
        A[i] ← 2A[i]/LW
    end for
    return D, A
end function
```

35
The algorithm shown is a bit simplified compared to the implement when it comes to the normalization step. The real implemented algorithm takes into account that nucleotides at the beginning and end of the gene does not get a score contribution from all sixty different window positions.

### 3.6 Measurement of performance (benchmark)

For monitoring the learning process and knowing when it is reasonable to stop the training, it is important to have a measurement of how well the neural network performs. This measurement process is also called *benchmarking*.

#### 3.6.1 Predicting splice sites

As mentioned earlier the transition from exon to intron is called a *donor splice site*. The algorithm for predicting exons and introns in the gene is more or less done as a finite state machine with two states – *exon* state and *intron* state. The gene sequence starts in *exon* state. The algorithm then searches for the first high value in the donor splice site indicator. When the algorithm finds a significant top in the donor splice site indicator, the state switches to *intron*. The algorithm continues to look for a significant top in the acceptor splice site indicator, and the state is switched back to *exon*. This process continues until the end of the gene. The gene must end in the *exon* state.

In the above paragraph, it is unclear what is meant by a *significant top*. To indicate a top in a splice site indicator, the algorithm first finds a indicator value above some threshold value. It then finds all successive indicator data points that are higher than this threshold value. Through all these values, a second order polynomial regression line is fitted, and the maximum of this parabola is used to indicate the splice site. This method is explained with some example data in Figure [3.4](#). In this example the indicator value at 0 and 1 is below the threshold. The value at 2 is just above the threshold and the successive values at 3,4,5 and 6 is also above the threshold and these five values are used in the curve fitting. The rest of the data points are below the threshold and not used in the curve fitting.

Finding a good threshold value is difficult. Several values have been tried. We have performed some simple experiments with dynamically computed threshold values based on average and standard deviation. However, the most practical threshold value is a constant at 0.2.
Figure 3.4: Predicting a splice site based on the splice site indicator. When the indicator reaches above the threshold value, 0.2 in the figure, all successive data points above this threshold are used in a curve fitting of a parabola. The nucleotide closest to the parabola maxima is used as the predicted splice site location.
Table 3.1: Four different outcomes when comparing the predicted exon/intron state and the actual state.

<table>
<thead>
<tr>
<th>Actual</th>
<th>Predicted</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Exon</td>
<td>Exon</td>
<td>True positive (TP)</td>
</tr>
<tr>
<td>Exon</td>
<td>Intron</td>
<td>False negative (FN)</td>
</tr>
<tr>
<td>Intron</td>
<td>Exon</td>
<td>False positive (FP)</td>
</tr>
<tr>
<td>Intron</td>
<td>Intron</td>
<td>True negative (TN)</td>
</tr>
</tbody>
</table>

3.6.2 Performance measurements

The above method is used to predict the locations of the exons and introns. These locations can be compared with the actual exon and intron locations. There are four different outcomes of this comparison. These outcomes are summarized in Table 3.1. The comparison of actual and predicted location is done at nucleotide level.

The counts of each comparison outcome are used to compute standard measurement indicators to benchmark the performance of the predictor. The sensitivity, specificity and correlation coefficient has been the de facto standard way of measuring the performance of prediction tools. These prediction measurement values are defined by Burset and Guigo [9] and by Snyder and Stormo [21].

**Sensitivity**

The sensitivity ($Sn$) is defined as the ratio of correctly predicted exon nucleotides to all actual exon nucleotides as given in Equation 3.3.

$$Sn = \frac{TP}{TP + FN} \quad (3.3)$$

The higher the ratio, the better prediction. As we can see, this ratio is between 0.0 and 1.0, where 1.0 is the best possible.

**Specificity**

The specificity ($Sp$) is defined as the ratio of correctly predicted exon nucleotides to all predicted exon nucleotides as given in Equation 3.4.

$$Sp = \frac{TP}{TP + FP} \quad (3.4)$$

The higher the ratio, the better prediction. As we can see, this ratio is between 0.0 and 1.0, where 1.0 is the best possible.
Correlation Coefficient

The correlation coefficient \( CC \) combines all the four possible outcomes into one value. The correlation coefficient is defined as given in Equation 3.5.

\[
CC = \frac{(TP \times TN) - (FN \times FP)}{\sqrt{(TP + FN)(TN + FP)(TP + FP)(TN + FN)}}
\]  

(3.5)

3.6.3 Reported performance measurements

When calculating performance measurements the algorithm report the sensitivity and specificity average of all genes in the data set. In addition the sensitivity, specificity and correlation coefficient for the whole dataset is calculated and reported.

The last reported value from the benchmarking is a simple error rate. It is calculated as given in Equation 3.6.

\[
\text{Error rate (err)} = \frac{FP + FN}{TP + FN + FP + TN}
\]  

(3.6)

This ratio is simply the number of incorrectly predicted nucleotides over all nucleotides. The lower ratio, the better. However this ratio is similar to the simple matching coefficient, \( SMC \), defined in Burset and Guigo [9]. This \( SMC \) is defined by Equation 3.7. The error rate then simply becomes \( 1 - SMC \).

\[
SMC = \frac{TP + TN}{TP + FN + FP + TN}
\]  

(3.7)

3.7 Overall training algorithm and implementation

Two tools has been developed. The training tool called \textit{trainer} and the benchmarking tool called \textit{benchmarker}. The \textit{trainer} tool can train a neural network from any specified data set. It will save a new neural network to the disc for each epoch through the dataset. The \textit{benchmarker} tool can only be used to benchmark a neural network on a specified data set. It can report sensitivities, specificity and correlation coefficient. It can report the average of all genes in the data set or it can be reported just for one specified gene. The tool can also export the indicator scores to plain text file.
3.7.1 GLib

The training system and benchmarking system was implemented in the C programming language, and the only non standard library that is used is GLib [10]. GLib is a "class" library for C, which provides basic data structures like linked lists, tree structures, hash tables, etc. It is the foundation library for the GTK+ toolkit system. In addition to the data structures, GLib also contains gobject and gtype. These provide the C programming language with an object system. With this object system it is possible to make object oriented code with known principles like inheritance, polymorphism and encapsulation. Even though the training systems do not take advantage of any object orientation, the neural network code that was used, was already implemented as a class in gobject. GLib also gives some convenient algorithms and data structures that makes development simpler and portability between different systems easy.

3.7.2 Neural network code

The neural network code used in this study is based on an earlier developed neural network which was used to learn to play backgammon. (See GNU Backgammon [24].) The neural network is now written as a class with the gobject[10] system. The very same neural network implementation was also used successfully in a study by Johansen [16] for training backgammon in combination with a k-means method.

3.7.3 The overall training algorithm

The main loop of the training is very simple and is an infinite loop with two significant activities. First, the infinite loop trains the neural network on all genes in the training data set. Second, it benchmarks the same neural network on the genes in the benchmark data set. An automatic way of breaking out of the main loop is simple to implement, but has not been done. Such break could be based on bootstrapping or a similar technique. There are also some other minor activities in the main loop like reshuffling the order of the training data set, saving the neural network, and logging the results. The main training loop is shown in Algorithm 4.

In this algorithm, $NN$ is the neural net to be trained, $T$ is the data set of genes to be used for training and $B$ is the data set of genes for benchmarking. In this algorithm the learning rate, $\eta$, is kept constant. The $bm$ variable is a composite data structure to hold the benchmarking data.

The subroutines SAVE() simply saves the neural network weights, and the
Algorithm 4 Main training loop

1: procedure Train(NN, T, B, η) \(\triangleright\) Train the neural network
2: repeat
3: for all \( g \in T \) do \(\triangleright\) Train neural net on each gene in dataset
4: \hspace{1em} TrainGene(NN, g, η)
5: end for
6: Save(NN)
7: Shuffle(T) \(\triangleright\) A new random order of the training set
8: for all \( g \in B \) do
9: \hspace{1em} Benchmark(NN, g, bm)
10: end for
11: LogResult(bm)
12: until break \(\triangleright\) Manually break when no improvement observed
13: end procedure

Shuffle() subroutine reorder the genes in the data set. The Shuffle() subroutine is based on pseudo code from Smed and Hakonen [20], and is listed in Appendix A.3. LogResult() simply logs the result to the terminal window and to a log file.

3.8 Summary of proposed method

The neural network is connected to the gene data structure by a sliding window. The inputs to the neural network are calculated by an orthogonal input scheme, and the output describes a score for where the splice sites are located in the sliding window. Evaluation of a gene is done by moving the window stepwise over the entire gene, and accumulates a splice site indicating score for each nucleotide.

The training is also performed by sliding a window over the entire gene, and using backpropagation to adjust the weights in the neural network. The desired output is computed from the score function.

For finding the exact location of the splice sites based on the evaluated indicator values, a second order polynomial function is fitted through the data points. The top of the parabola is used as the predicted location of the splice site.

The performance measurement has been defined, and the overall training algorithm is listed as pseudo code. In the next chapter we will describe some of the experiments and the results of the experiments.
Chapter 4

Training and results

In this chapter we present the experiments performed and the results. The first part presents an initial test performed to verify that the system worked properly and that the proposed algorithms are able to predict splice sites. The last part presents experiments where the training data set of 15551 genes were used. The results are discussed and improvements are suggested.

4.1 Initial testing

After the code completion, the first basic test is done to see if the method is usable at all. Ten genes were selected more or less at random. The initial test was to train a neural net on these ten genes and then benchmark this system on the same ten genes. If this does not work, the method can not be really good at all. The ten selected genes are listed in Table 4.1.

A neural network was generated with random weights. This neural network was then trained for 1000 epochs\(^1\) with a constant learning rate of 0.5.

The result of this simple experiment was promising. The correlation coefficient showed 0.9524 for this test. The full result is showed in Table 4.2. The numbers are really good, but the neural network was presented about

\[^1\text{An epoch is one run through the dataset of training data}\]

Table 4.1: The ten genes selected for the initial test.

<table>
<thead>
<tr>
<th>AT1G31380.1</th>
<th>AT1G03340.1</th>
<th>AT1G04440.1</th>
<th>AT1G31390.1</th>
<th>AT3G10680.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT3G20900.1</td>
<td>AT2G12695.1</td>
<td>AT3G11500.1</td>
<td>AT2G27700.1</td>
<td>AT1G05170.2</td>
</tr>
</tbody>
</table>
Table 4.2: Results for the initial test where a neural network were specially trained on ten genes, and how it evaluates again on the same ten genes.

<table>
<thead>
<tr>
<th>Measure</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average of the 10 genes</td>
<td></td>
</tr>
<tr>
<td>Sensitivity (Sn)</td>
<td>0.9947</td>
</tr>
<tr>
<td>Specificity (Sp)</td>
<td>0.9623</td>
</tr>
<tr>
<td>Measures of total sequence</td>
<td></td>
</tr>
<tr>
<td>Sensitivity (Sn)</td>
<td>0.9958</td>
</tr>
<tr>
<td>Specificity (Sp)</td>
<td>0.9777</td>
</tr>
<tr>
<td>Correlation (CC)</td>
<td>0.9614</td>
</tr>
</tbody>
</table>

16000 training patterns, and a neural network of 31106 adjustable weights, should be able to learn this well.

In Figure 4.1, the splice site indicator curves has been plotted for the entire ten genes. The red curve is the donor splice site indicator, and the green curve is the acceptor splice site indicator. The lines above the indicators, shows the actual and predicted exons. The upper line (magenta) indicates the predicted exons and the other line (blue) indicated the actual exons. With the exception of gene AT3G20900.1, it looks like the indicators perfectly find splice sites. The parabola method of addressing the location also seems to work fine. There are only some small mismatches due to the low-pass filtering effect of the sliding window. For the case of AT3G20900.1 it looks like it is more or less luck that the parabola method finds anything at all.

4.2 Extensive training sessions

The training data set of 15551 genes where then used to train a neural network. The training was done in three sessions, and for each session we chose separate, but constant, learning rates. The learning rate, $\eta$, was chosen to be 0.5, 0.25, and 0.1, respectively. For each epoch through the training data set, the neural network performance was measured with the benchmark data set.

One epoch through the data set of 15551 genes takes about 20 minutes to complete, depending on the hardware.

The idea of the benchmarking for each epoch was to be able to monitor the development of the training and hopefully observe that the error rate goes down as the training prevails. A plot of the error rate as a function of the number of epochs is showed in Figure 4.2. The plot only shows the error
Figure 4.1: The genes in the initial test. In each figure the donor splice site indicator is red, and the acceptor splice site indicator is plotted green. Above each of the graphs are the actual and predicted exons. The upper line indicates the predicted exon and the other line indicate the actual exon.
rate for the first 80 epochs of training, and the learning rate in this training session was 0.5. The figure shows a correlation of the number of epochs and the error rate. The correlation was calculated to -0.53. The correlation between error rate and the number of epochs trained seems to work good for the first 80 epochs. The figure indicates that the error rate has a high variance.

If the figure is extended to 168 epochs, it seems like the correlation ceases. See Figure 4.2. The correlation has not been calculated, but in the figure it looks like the error rate increases after the first 80 epochs. The reason for this can be that the learning rate, $\eta$, is too high, and that this high value disturbs the training.

### 4.2.1 Training session with learning rate 0.5

This was the initial training session. It was started from a neural network based on random weights. The random weights were randomly selected in the range for -10.0 to 10.0, however weight values in the range -0.1 to 0.1 were rejected. This session was running for 168 epochs. The best neural network from this session achieved a $CC$ of 0.5616. For one single gene, AT1G31380.1, the splice site indicators were calculated for each epoch trained. The indicator peak amplitude seems to fluctuate for each epoch. This particular gene is good for showing this effect, since it has only one intron, and therefore only
one donor splice site and one acceptor splice site. The gene was chosen from the benchmark data set. In Figure 4.4 the maximum value of the splice site indicators are plotted for each epoch in the training session. As expected, maximum value increases for the first 80 epochs. However, after these first 80 epochs the maximum value seems to fluctuate even more and not converge towards 1.0 at all. These fluctuations can be interpreted as the learning rate being too high. The learning rate was therefore reduced to 0.25.

4.2.2 Training session with learning rate 0.25

A new training session with a learning rate of 0.25 were performed. The training started from the best neural network, (best according to SMC), from the previous session. This training session was kept running for 1753 epochs. Unfortunately no correlation of the performance measurements and the number of trained epoch could be observed. In the same way the fluctuations of the maximum splice site indicator value is plotted for each epoch. The fluctuations are not quantified by statistical indicators. These fluctuations can indicate that the learning rate is still too high.

The best neural network with respect to the correlation coefficient, achieved a $CC$ of 0.5679. This is just marginally better than the best neural network from the session with learning rate 0.5.
Figure 4.4: Fluctuations of the maximum splice site indicator in AT1G31380.1 for the training session using a learning rate of 0.5. The maximum value is plotted for each epoch.
Figure 4.5: Fluctuations of the maximum splice site indicator in AT1G31380.1 for the training session using a learning rate of 0.25. The maximum value is plotted for each epoch. Only the donor splice site maximum is shown in this figure.
4.2.3 Training session with learning rate 0.1

A training session with a learning rate of 0.1 was also performed. This training was started from a neural network that had a $CC$ of 0.3328, and was the latest neural network from the training session using $\eta = 0.25$ available at the time the training session initiated. The training session was stopped after 863 epochs.

Unfortunately this training session did not improve the neural network. A plot of the error rate for each epoch is shown in Figure 4.6. No correlation between the number of epochs and the error rate can be observed. This may indicate that the learning rate is still too high, and that the high learning rate disturbs the training. A high learning rate changes the weights in the neural network such that the network gets out of a minimum and diverges. The weights changes so much that the adjustments acts like noise to the neural network.

The neural network with the best measured performance from this session, achieved a $CC$ of 0.5134, which is worse than the best neural network from the previous network.
4.3 Finding splice sites in a particular gene

The splice site indicators can be plotted for a single gene. To illustrate our results, we present an arbitrarily chosen gene, AT3G11730.1. The curves in Figure 4.7 represent the donor and acceptor splice site indicators for the entire gene. The donor splice sites are marked using a red line, the acceptor splice sites using a green line, and the predicted and actual exons are marked with the upper and lower dashed lines, respectively. The shown indicators are computed using a neural network which has been trained for about 1400 epochs, with a learning rate of 0.25. As noted in the header of Figure 4.7, the prediction on this gene achieves a better than average $CC$ of 0.857. We can also point out that the neural network has erroneously missed an intron in the region covered by nucleotides 300 to 400. It is clear that acceptor splice site would have been found in this case, but due to the corresponding donor splice site was not found, the whole intron was erroneously missed. We can also point out the false top in the donor splice site indicator at 1200. This top would clearly have indicated a false donor splice site, but this top is correctly ignored, because there is no corresponding acceptor splice site. Comparing the missed top in the donor splice site located at about 300, and the false top in the same curve located at about 1200, we see that the false top is significantly higher than the missed true top. Observations like this indicate that a general system to predict the exon and intron locations based on the indicator values can be difficult to develop.

Even though we missed an intron here, the results are promising. Most other splice sites match the actual data, and some of the errors are most likely due to the low-pass filtering effect of using a sliding window, causing ambiguous splice sites.

4.4 Measurements of the best neural networks

The best performing neural network, achieved a correlation coefficient of 0.568. The correlation coefficients, as well as the sensitivity, specificity, and standard simple matching coefficient ($SMC$), are shown in Table 4.3. When calculating these performance measurements, the benchmark algorithm averages the sensitivities and specificities for all genes in the data set. In addition the specificity, sensitivity, and correlation coefficient for the entire dataset is reported.
Figure 4.7: The splice site indicators plotted along an arbitrary gene (AT3G11730.1). Above the splice site indicators, there are two line indicators where the upper line indicates predicted exons, and the other line indicates actual exons. The sensitivity, specificity and correlation coefficient of this gene is given in the figure heading. (*Err* is an error rate defined as the ratio of false predicted nucleotides to all nucleotides. *Err* = 1 − *SMC*.)

Table 4.3: Performance measurements of the best neural network performances for each of the three training sessions.

<table>
<thead>
<tr>
<th>Session</th>
<th>Average</th>
<th>All nucleotides in set</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><em>Sn</em></td>
</tr>
<tr>
<td>η = 0.50</td>
<td>0.857</td>
<td>0.867</td>
</tr>
<tr>
<td>η = 0.25</td>
<td>0.874</td>
<td>0.857</td>
</tr>
<tr>
<td>η = 0.10</td>
<td>0.743</td>
<td>0.894</td>
</tr>
</tbody>
</table>
Chapter 5

Discussion and conclusion

5.1 Conclusion

This study shows an artificial neural networks used in splice site prediction. The best neural network trained in this study, achieve a correlation coefficient at 0.568. This result is achieved without prior knowledge of any sensor signals, like ‘GT’ or ‘GC’ for the donor splice sites, or ‘AG’ for the acceptor splice sites. Also note that some of the genes in the data sets did not store the base case for splicing, but an alternative splicing, which may have disturbed some of the training. It is fair to conclude that artificial neural networks are usable in gene prediction, and the method used, with a sliding window over the gene, is worth further study. This method combined with other statistical methods, like General Hidden Markov Models, would probably improve the results further.

5.2 Suggested further work

This thesis is only a start of what can be done with neural networks for predicting genes and splice sites in DNA sequences. The following list mentions some ideas that can be further examined.

Cleanup in data sets. The operation of extracting genes from the data sets was done more or less automatically. One of the unfortunate things that happened in this cleaning process was the loss of genes with alternative splicing. If a gene had an alternative splice, only the splicing variant with highest numeral value where kept. Other splice variations excluded, and therefore also the splicing base case.

Improve the overall training algorithm. The overall training algorithm
uses a constant learning rate. Some of the results indicate that the learning rate is too high. The neural network can be retrained with a lower learning rate. An algorithm that adjusts the learning rate automatically could also be considered.

**Analysing the predicted splice sites.** Is it possible to see some kind of pattern in those splice sites that the neural network finds, and those that is not found? This can be checked by finding common substrings or running an expectation maximisation algorithm on the sequences where slice sites where found.

**Analysing the neural network inputs.** Maybe some of the neural network inputs do not contribute to the learning. Some experiments could be performed to check if the system performs better or worse with a smaller sliding window. And of course also check what happens if the sliding window is set wider. More inputs, other than the orthogonal inputs, can be added to the neural network. This could be other properties, like 'G' and 'C' content in the window, mean number of each base or other properties.

**Find effect of number of hidden units.** In this study, there were 128 hidden units in the neural network. A bigger neural network with more hidden units may be able to learn better. Also, the more weights to adjust, the longer time must be used for training. Maybe even a smaller neural network can perform just as good, and with less training.

**Find alternative methods for finding indicator peaks.** It is possible to develop other method for finding slice sites based on the indicator value. The current scheme of finding the splice sites is based on finding the maximum of a parabola which is curve fitted through some of the data points. This may be improved. Several things have been discussed. Filtering the indicator values with a FIR filter can be done. A watershedding algorithm has also been suggested. None of these has been tried. The splice site indicators could also be analysed together, such that if the system can finds two certain donor splice sites, it must find an acceptor splice site in-between these. Searching for special dinucleotide patterns like 'GT', 'GC' and 'AG' can also be considered.

**Combine with other methods.** The developed method could of course be combined with other statistical methods. A combination of several methods can be done in several ways. Most natural would be to run a statistical method that indicates the genes first, and then run this
neural network method on the predicted genes. However, an other statistical model can also be used for post-processing the results for the neural network evaluations.

On this list, the two first items are trivial, and may also be the most important. If the data that the system is training and benchmarking on is not correct, the results will be worse.

The other items on this list require more research and ideas will have to be tested and analysed.

There are also some code optimizations that can be done. No profiling has been performed on the running code, and some algorithms may perform unnecessarily slow. For instance, it is possible to improve the calculation of the desired output in the training phase. The splices sites in the window are just moved one step in the sliding window, so the desired output could be calculated based on the previous output when the window was at its previous position.
Appendix A

Additional pseudo code

A.1 Feedforward calculation

In algorithm 5 the feed forward calculation is listed. The functions input parameter \( NN \) is a composite data structure that contains the weight matrices \( W_1 \) and \( W_2 \) and the bias vectors \( b_1 \) and \( b_2 \). The algorithm is stated as it would be written in a programming language that can do fundamental matrix operations on a matrix data type.

**Algorithm 5 Feedforward operation**

1: **function** Evaluate(\( NN, x \)) $\triangleright$ Forward evaluation of input vector \( x \)
2: **in:** A neural network \( (NN) \), a input vector \( x \)
3: **out:** A output vector \( z \)
4: \( \text{net}_1 \leftarrow W_1x + b_1 \)
5: \( y \leftarrow f(\text{net}_1) \)
6: \( \text{net}_2 \leftarrow W_2y + b_2 \)
7: \( z \leftarrow f(\text{net}_2) \)
8: **return** \( z \)
9: **end function**

A little trick can be done when this function is called by the backpropagation algorithm. The backpropagation algorithms can use the \( y \) vector to optimize the sensitivity calculation. A real implementation should then return both the \( y \) vector and the \( z \) vector.
### A.2 Backpropagation

The backpropagation calculation is listed in Algorithm 6. As in the EVALUATE() function, this is listed as it would in a programming language the do fundamental matrix operations on a matrix data type. The neural net (NN) input parameter is a composite data structure that contains $W_1$, $W_2$, $b_1$ and $b_2$.

**Algorithm 6 Backpropagation**

1: procedure BACKPROPAGATION($NN, x, t, \eta$)  \(\triangleright\) Adjust weights
2: in: A neural net ($NN$), a input vector $x$, a desired output $t$, a learning rate $\eta$
3: out: None, weights of $NN$ gets adjusted.
4: \(z, y\) ← EVALUATE($NN, x$)  \(\triangleright\) return both $z$ and $y$
5: $\delta_2 ← -z(1 - z)(t - z)$  \(\triangleright\) Sensitivity
6: $\delta_1 ← -y(1 - y)W_2\delta_2$  \(\triangleright\) Sensitivity
7: $\Delta W_2 ← \eta\delta_2 y^T$
8: $\Delta W_1 ← \eta\delta_1 x^T$
9: $W_2 ← W_2 + \Delta W_2$
10: $W_1 ← W_1 + \Delta W_1$
11: $b_2 ← b_2 + (\eta\delta_2)$
12: $b_1 ← b_1 + (\eta\delta_1)$
13: end procedure

Note that the sensitivity calculation in lines 5 and 6 are depending on the activation function. The above algorithm assumes an activation function as shown in Equation 3.1 with a $\beta$ of -1.

### A.3 Shuffle

The SHUFFLE algorithm (See Algorithm 7) takes in a ordered set, and randomly shuffles the elements. This pseudo code and the implemented algorithms modify the list in-place. The code is based on algorithm 2.5 from Smed and Hakonen [20].

The RANDOM-INTEGER() function returns a random integer in the range given by the two input parameters.
Algorithm 7 Random Shuffle

1: **procedure** SHUFFLE($R$)
2: **in:** ordered set $R$
3: **out:** None. The set is ordered in-place.
4: \hspace{1em} for $i \leftarrow 0$ to $\text{length}[R] - 2$ do
5: \hspace{2em} $j \leftarrow \text{RANDOM-INTEGER}(i, \text{length}[R])$
6: \hspace{2em} swap $R[i] \leftrightarrow R[j]$
7: end for
8: end procedure
Appendix B

Excluded genes

Here follows a description of all excluded genes in training set. Some genes may have several reasons to be excluded, but all excluded genes are only listed once.

B.1 Single exon genes

This is 10475 genes in total. The value of such listing is limited. The list is therefore omitted for practical purposes. The list can be provided on request.

B.2 Unknown nucleotides

AT1G39430.1 Contains N
AT2G48110.1 Contains M
AT2G08986.1 Contains M
AT3G04670.1 Contains M
AT3G04670.2 Contains M
AT1G33610.1 Contains s

B.3 Short exons or introns

There is 1070 genes where at least one exon is only 30 nucleotides or less. In addition 94 genes contains short introns. This is so long that it is impractical to list all these genes. The list can be provided on request.
Appendix C

Codons to amino acids

The translation translates codons to the following amino acids.

<table>
<thead>
<tr>
<th>Amino Acid</th>
<th>Codon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ala</td>
<td>GCA GGC GCG GCT</td>
</tr>
<tr>
<td>Cys</td>
<td>TGC TGT</td>
</tr>
<tr>
<td>Asp</td>
<td>GAC GAT</td>
</tr>
<tr>
<td>Glu</td>
<td>GAA GAG</td>
</tr>
<tr>
<td>Phe</td>
<td>TTC TTT</td>
</tr>
<tr>
<td>Gly</td>
<td>GGA GGC GG GGT</td>
</tr>
<tr>
<td>His</td>
<td>CAC CAT</td>
</tr>
<tr>
<td>Ile</td>
<td>ATA ATC ATT</td>
</tr>
<tr>
<td>Lys</td>
<td>AAA AAG</td>
</tr>
<tr>
<td>Leu</td>
<td>TTA TTG CTA CTC CTG CTT</td>
</tr>
<tr>
<td>Met</td>
<td>ATG</td>
</tr>
<tr>
<td>Asn</td>
<td>AAC AAT</td>
</tr>
<tr>
<td>Pro</td>
<td>CCA CCC CCG CCT</td>
</tr>
<tr>
<td>Glu</td>
<td>CAA CAG</td>
</tr>
<tr>
<td>Arg</td>
<td>AGA AGG CGA CGC CGG CGT</td>
</tr>
<tr>
<td>Ser</td>
<td>AGC AGT TCA TCC TCG TCT</td>
</tr>
<tr>
<td>Thr</td>
<td>ACA ACC ACG ACT</td>
</tr>
<tr>
<td>Val</td>
<td>GTA GTC GTG GTT</td>
</tr>
<tr>
<td>Trp</td>
<td>TGG</td>
</tr>
<tr>
<td>Tyr</td>
<td>TAC TAT</td>
</tr>
</tbody>
</table>

Table C.1: Translation from codons to amino acids
Appendix D

Submitted article

The following article has been submitted to Fifth International Meeting on Computational Intelligence Methods for Bioinformatics and Biostatistics, CIBB 2008. This conference will be held 3–4 October 2008, in Vietri sul Mare, Salerno in Italy. http://cibb08.disi.unige.it/.
SPLICE SITE PREDICTION USING ARTIFICIAL NEURAL NETWORKS

Øystein Johansen, Tom Ryen, Trygve Eftestøl, Thomas Kjosmoen, Peter Ruoff

University of Stavanger
Address: 4036 Stavanger, Norway, email: o.johansen@stud.uis.no, telephone: +4795142798

Keywords: Splice site prediction, Artificial neural network, Arabidopsis thaliana, Sliding window, Orthogonal input representation.

Abstract. A system for utilizing an artificial neural network to predict splice sites in genes has been studied. The neural network uses a sliding window of nucleotides over a gene and predicts possible splice sites. Based on the neural network output, the exact location of the splice site is found using a curve fitting of a parabolic function. The splice site location is predicted without prior knowledge of any sensor signals, like ’GT’ or ’GC’ for the donor splice sites, or ’AG’ for the acceptor splice sites. The neural network has been trained using backpropagation on a set of 15551 genes of the model plant Arabidopsis thaliana. The performance is then measured using a completely distinct gene set of 5000 genes. The best measured performance on the test data set gives a sensitivity of 0.851, a specificity of 0.844 and a correlation coefficient of 0.568.

1 Introduction

Gene prediction has become more and more important as the DNA of more organisms are sequenced. DNA sequences submitted to databases are often already characterized and mapped when they are submitted. This means that a molecular biologist has already used genetics and biochemical methods to find genes, promoters, exons and other meaningful subsequences in the submitted material. However, the number of sequencing projects are increasing, and a lot of DNA sequences have not yet been mapped or characterized. Having a computational tool to predict genes and other meaningful subsequences is therefore of great value, and can save a lot of expensive and time consuming experiments for biologists.

This study tries to utilize an artificial neural network to predict where the splice sites of a gene can be located. The splice sites are the transitions from exon to intron or from intron to exon. A transition from exon to intron is called a donor splice site and a transition from intron to exon is called acceptor splice site.

2 Neural network

The main premise in this study is to use a window of nucleotides that moves stepwise over the sequence to be analysed. The inputs to the neural network are calculated from the input calculator. For each step of the sliding window the neural network will give an output score if it recognizes there is a splice site in the window. A diagram of the entire prediction system is shown in Fig. 1. The window size is chosen to be 60 nucleotides. This is hopefully wide enough to find significant patterns on both sides of the splice site. A bigger window will make the neural network bigger and thereby harder to train. Smaller window would maybe exclude important information around the splice site.
2.1 Network topology

The neural network structure is a standard three layer feedforward neural network. There are two output units corresponding to the donor and acceptor splice sites, 128 hidden layer units and 240 input units. The 240 input units were used since the orthogonal input scheme uses four inputs each nucleotide in the window. The number of hidden layer units were chosen based on a previous work by Johansen [6], that showed that neural network with this size were able learn complex concepts. Also, the neural network program code was reused from that study, and the code was optimized for this number of hidden units. There is also a bias signal added to the hidden layer and the output layer. The total number of adjustable weights is therefore $(240 \times 128) + (128 \times 2) + 128 + 2 = 31106$.

2.2 Activation function

The activation function is a standard sigmoid function, shown in Eq. 1. The $\beta$ values for the sigmoid functions are 0.1 for both the hidden layer activation and the output layer activation. Preliminary experiments were performed to test the effect of these values. These tests indicated that 0.1 was a suitable value. Previous work by Johansen [6] has shown that the effects of the $\beta$ values are negligible with respect to how fast the weights converge.

$$f(x) = \frac{1}{1 + e^{-\beta x}}$$  \hspace{1cm} (1)

When doing forward calculations and backpropagation, the sigmoid function is called repetitively. It is therefore important that this function has a high computational performance. A fast and effective evaluation of the sigmoid function can improve the overall performance considerably. To

---

1This kind of neural network has several names, such as multilayer perceptrons (MLP), feed forward neural network, and backpropagation neural network.
improve the performance of the sigmoid function, a precalculated table for the exponential function is used. This table lookup method is known to be very fast, and with an acceptable accuracy.

2.3 Backpropagation

The neural network was trained using a normal backpropagation method as described in Duda, Hart, Stork [3], Haykin [4] or Kartalopoulos [7]. There is no momentum used in the training. We have not implemented any second order methods to help the convergence of the weights.

3 Training data and benchmarking data

Based on data from the The Arabidopsis Information Resource (TAIR) release 7 website [9], we compiled a certain set of genes. TAIR is an on-line database resource of genetic and molecular biology data of the model plant Arabidopsis thaliana.

3.1 Excluded genes

All genes that contain unknown nucleotides were excluded from the data set. In addition, all single exon genes were excluded. Further, all genes with very short exons or introns were excluded. By ”short” we mean 30 nucleotides or less. These genes were excluded to avoid very short exons or very short introns. Excluding these genes also simplifies the calculation of desired outputs, since it then can not be more than two splice sites in a window.

For genes with alternative splicing, only one splicing variation was kept. The only kept splice variation was the one with the highest numeral value. For instance, AT5G58140.4 was included in the data set, but AT5G58140.1, AT5G58140.2 and AT5G58140.3 were excluded.

3.2 Training data set and benchmark data set

The remaining data set, after exclusion of some genes, consists of 20551 genes. This set is divided into a training data set and a benchmarking data set. The training set and the benchmark set have 15551 and 5000 genes, respectively. The number of genes in each set is chosen such that the benchmark set is large enough to achieve a reliable performance measure of the neural network. This splitting was done at random. Both data sets contains genes from all five chromosomes.

4 Training method

The neural network training is done using standard backpropagation. This section describes how the neural network inputs were calculated and how the desired output was obtained.

4.1 Sliding window

For each gene in the training set, we let a window slide over the nucleotides. The window moves one nucleotide each step, covering a total of \( L_G - L_W + 1 \) steps, where \( L_G \) is the length of the gene and \( L_W \) is the length of the window. As mentioned earlier, the length of the window is 60 nucleotides in this study.

4.2 Input to the neural network

For each nucleotide in the sliding window, we have four inputs to the neural network. The four inputs are represented as an orthogonal binary vector. (A=1000, T=0100, G=0010, C=0001). This input description has been used in several other studies [5], and is described in Baldi and Brunak [1]. This input system is called orthogonal input, due to the orthogonal binary vectors. According to Baldi and Brunak [1] this is the most used input representation for neural networks in the application of gene prediction. This input scheme also has the advantage that each nucleotide input is separated from each other, such that no arithmetic correlation between the monomers need to be constructed.
4.3 Desired output and scoring function

The task is to predict splice sites, thus the desired output is 1.0 when a splice site is in the middle of the sliding window. There are two outputs from the neural network; one for indicating acceptor splice site and one for indicating donor splice site.

However, if it is only a 1.0 output when a splice site is in the middle of the window and 0.0 when a splice site is not in the middle of the window, there will probably be too many 0.0 training samples that the neural network would learn to predict everything as ‘no splice site’. This is why we introduce a score function which calculates a target output not only when the splice site is in the middle of the window, but whenever there is a splice site somewhere in the window. We use a weighting function where the weight of a splice site depends on the distance from the respective nucleotide to the nucleotide at the window mid-point. The further from the mid point of the window this splice site is, the lower value we get in the target values. The target values decrease linearly from the mid point of the window. This gives the score function as shown in Eq. 2

\[
f(n) = 1 - |1 - \frac{2n}{L_W}|
\]  

Eq. 2

If a splice site is exactly at the mid point, the target output is 1.0. An example window is shown in Fig. 2.

In some cases there may be two splice sites in a window. It is then one acceptor splice site and one donor splice site.

4.4 Algorithm for training on one single gene

The algorithm for training on a single gene is very simple. The program loops through all the possible window positions. For each position, the program computes the desired output for the current window, computes the neural network input and then calls BACKPROPAGATION() with these computed data. See Algorithm 1.

In this algorithm, NN is a composite data structure that holds the neural network data to be trained. G is one specific gene that the neural network should be trained on. The first integer variable n is simply the total number of positions of the sliding window. L_W is the number of nucleotides in the sliding window, which in this study, is set to 60 nucleotides.

The desired output is calculated as described in Section 4.3, and is listed in Algorithm 2.

5 Evaluation method

The evaluation of a gene is simply the forward calculation performed for all the window positions in that gene. The neural network outputs are accumulated as an indicator value for each
Algorithm 1 Training the neural net on one single gene

```
1: procedure TRAINGENE(NN, G, η) ⊸ Train the network on gene G
2:     n ← length[G] − LW
3:     for i ← 0 to n do ⊸ Slice the gene sequence
4:         W ← G[i..(i + LW)]
5:         desired ← CALCULATEDESIRED(W)
6:         input ← CALCULATEINPUT(W)
7:         BACKPROPAGATION(NN, input, desired, η)
8:     end for
9: end procedure
```

Algorithm 2 Calculating the desired output based on a given window

```
1: function CALCULATEDESIRED(W) ⊸ Calculate desired output
2:     D ← [0.0, 0.0] ⊸ Initialize the return array
3:     prev ← ISEXON(W[0]) ⊸ Boolean value
4:     if prev then
5:         j ← 0
6:     else
7:         j ← 1
8:     end if
9:     for i ← 1 to LW do
10:        this ← ISEXON(W[i]) ⊸ Score function
11:        if prev ≠ this then
12:           D[j] ← D[j] + 1 − |1 − 1/30| ⊸ Flip the index 0 to 1, 1 to 0
13:           j ← 1 − j
14:        end if
15:     prev ← this
16:     end for
17: return D
18: end function
```
nucleotide in the gene.

5.1 Sliding window
Because the neural network is trained to recognize splice sites in a 60 nucleotides wide window, the forward calculation process is also performed on the same sized window. The window slides over the gene in the same way as in the training procedure.

5.2 Cumulative output and normalization
The sliding window moves over the gene and forward calculates whenever there is a splice site in the window. A nucleotide gets a score contribution from 60 outputs corresponding to the sliding window passing over it. All these outputs are accumulated.

The accumulated output is then normalized. Most of the nucleotides will get a contribution from 60 different window positions, and these nucleotides are normalized by dividing the cumulative output by the area under the score function (30.0). These normalized cumulative scores are called acceptor splice site indicator and donor splice site indicator.

5.3 Algorithm for evaluating one single gene
The pseudo code of the evaluation of a gene is given in Algorithm 3. The algorithm contains two loops. The first loop a slides a window over all positions in the gene and adds up all the predictions from the neural network. The second loop normalizes the splice site indicators.

Algorithm 3 Evaluation of gene

```
function EVALUATEGENE(NN, G) ▷ Calculate splice site indicators
  in: Neural network (NN), Gene (G)
  out: Two arrays, D and A, which contains the donor and acceptor splice site indicator.
  n ← length[G] − LW
  for i ← 0 to n do
    W ← G[i..(i + LW)] ▷ Slice the gene sequence
    input ← CALCULATEINPUT(W)
    pred ← EVALUATE(NN, input) ▷ Gets predicted output
    for j ← i to LW + i do
      D[j] ← D[j] + pred[0]
    end for
  end for
  for i ← 0 to length[G] − 1 do ▷ Normalizing loop
    D[i] ← 2D[i]/LW
    A[i] ← 2A[i]/LW
  end for
  return D, A
end function
```

6 Measurement of performance (benchmark)
For monitoring the learning process and knowing when it is reasonable to stop the training, it is important to have a measurement of how well the neural network performs. This measurement process is also called benchmarking.
6.1 Predicting splice sites

As mentioned earlier the transition from exon to intron is called a donor splice site. The algorithm for predicting exons and introns in the gene is more or less done as a finite state machine with two states – exon state and intron state. The gene sequence starts in exon state. The algorithm then searches for the first high value in the donor splice site indicator. When the algorithm finds a significant top in the donor splice site indicator, the state switches to intron. The algorithm continues to look for a significant top in the acceptor splice site indicator, and the state is switched back to exon. This process continues until the end of the gene. The gene must end in the exon state.

In the above paragraph, it is unclear what is meant by a significant top. To indicate a top in a splice site indicator, the algorithm first finds an indicator value above some threshold value. It then finds all successive indicator data points that are higher than this threshold value. Through all these values, a second order polynomial regression line is fitted, and the maximum of this parabola is used to indicate the splice site. This method is explained with some example data in Fig. 3. In this example the indicator value at 0 and 1 is below the threshold. The value at 2 is just above the threshold and the successive values at 3, 4, 5 and 6 is also above the threshold and these five values are used in the curve fitting. The rest of the data points are below the threshold and not used in the curve fitting.

Finding a good threshold value is difficult. Several values have been tried. We have performed some simple experiments with dynamically computed threshold values based on average and standard deviation. However, the most practical threshold value was found to be a constant at 0.2.

6.2 Performance measurements

The above method is used to predict the locations of the exons and introns. These locations can be compared with the actual exon and intron locations. There are four different outcomes of this comparison. These outcomes are summarized in Tab. 1. The comparison of actual and predicted location is done at nucleotide level. The count of each comparison outcome are used to compute standard measurement indicators to benchmark the performance of the predictor. The sensitivity, specificity and correlation coefficient has been the de facto standard way of measuring the performance of prediction tools. These prediction measurement values are defined by Burset and Guigó [2] and by Snyder and Stormo [8].
### Table 1: Four different outcomes when comparing the predicted exon/intron state and the actual state

<table>
<thead>
<tr>
<th>Actual</th>
<th>Predicted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exon</td>
<td>Exon</td>
</tr>
<tr>
<td>Exon</td>
<td>Intron</td>
</tr>
<tr>
<td>Intron</td>
<td>Exon</td>
</tr>
<tr>
<td>Intron</td>
<td>Intron</td>
</tr>
</tbody>
</table>

The sensitivity ($Sn$) is defined as the ratio of correctly predicted exon nucleotides to all actual exon nucleotides as given in Eq. 3.

$$Sn = \frac{TP}{TP + FN} \quad (3)$$

The higher the ratio, the better prediction. As we can see, this ratio is between 0.0 and 1.0, where 1.0 is the best possible.

The specificity ($Sp$) is defined as the ratio of correctly predicted exon nucleotides to all predicted exon nucleotides as given in Eq. 4.

$$Sp = \frac{TP}{TP + FP} \quad (4)$$

The higher the ratio, the better prediction. As we can see, this ratio is between 0.0 and 1.0, where 1.0 is the best possible.

The correlation coefficient ($CC$) combines all the four possible outcomes into one value. The correlation coefficient is defined as given in Eq. 5.

$$CC = \frac{(TP \times TN) - (FN \times FP)}{\sqrt{(TP + FN)(TN + FP)(TP + FP)(TN + FN)}} \quad (5)$$

### 6.3 The overall training algorithm

The main loop of the training is very simple and is an infinite loop with two significant activities. First, the infinite loop trains the neural network on all genes in the training data set. Second, it benchmarks the same neural network on the genes in the benchmark data set. There are also some other minor activities in the main loop like reshuffling the order of the training data set, saving the neural network, and logging the results. The main training loop is shown in Algorithm 4.

```
Algorithm 4 Main training loop
1: procedure TRAIN(NN, T, B, η) \> Train the neural network
2: repeat
3: \> Train neural net on each gene in dataset
4: \> for all $g \in T$ do
5: \> \> TRAINGENE(NN, g, η)
6: \> end for
7: \> SAVE(NN)
8: \> SHUFFLE(T) \> A new random order of the training set
9: \> for all $g \in B$ do
10: \> \> BENCHMARK(NN, g, bm)
11: \> end for
12: \> LOGRESULT(bm) \> Manually break when no improvement observed
13: until break
14: end procedure
```
In this algorithm, $NN$ is the neural net to be trained, $T$ is the data set of genes to be used for training and $B$ is the data set of genes for benchmarking. In this algorithm the learning rate, $\eta$, is kept constant. The $bm$ variable is a composite data structure to hold the benchmarking data.

The subroutine SAVE() saves the neural network weights, and the SHUFFLE() subroutine reorders the genes in the data set. LOGRESULT() logs the result to the terminal window and to a log file.

7 Experiments and results

The training data set of 15551 genes where then used to train a neural network. The training was done in three sessions, and for each session we chose separate, but constant, learning rates. The learning rate, $\eta$, was chosen to be 0.5, 0.25, and 0.1, respectively. For each epoch² through the training data set, the neural networks performance was measured with the benchmark data set.

7.1 Finding splice sites in a particular gene

The splice site indicators can be plotted for a single gene. To illustrate our results, we present an arbitrarily chosen gene, AT3G11730.1. The curves in Fig. 4 represent the donor and acceptor splice site indicators for an entire gene. The donor splice sites are marked using a red line, the acceptor splice sites using a green line, and the predicted and actual exons are marked with the upper and lower dashed lines, respectively. The shown indicators are computed using a neural network which has been trained for about 1400 epochs, with a learning rate of 0.25. As noted in the header of Fig. 4, the prediction on this gene achieves a better than average $CC$ of 0.857. We can also point out that the neural network has erroneously missed an intron in the region covered by nucleotides 300 to 400. Even though we missed an intron here, the results are promising. Most other splice sites match the actual data, and some of the errors are most likely due to the low-pass filtering effect of using a sliding window, causing ambiguous splice sites.

7.2 Measurements of the best neural networks

The best performing neural network, achieved a correlation coefficient of 0.568. The correlation coefficients, as well as the sensitivity, specificity, and standard simple matching coefficient ($SMC$), are shown in Tab. 2. When calculating these performance measurements, the benchmark algorithm averages the sensitivities and specificities for all genes in the data set. In addition the specificity, sensitivity, and correlation coefficient for the entire dataset is reported.

²An epoch is one run through the data set of training data.
Table 2: Performance measurements of the best neural network performances for each of the three training sessions.

<table>
<thead>
<tr>
<th>Session</th>
<th>Average</th>
<th></th>
<th>All nucleotides in set</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sn</td>
<td>Sp</td>
<td>Sn</td>
<td>Sp</td>
</tr>
<tr>
<td>$\eta = 0.50$</td>
<td>0.857</td>
<td>0.867</td>
<td>0.825</td>
<td>0.854</td>
</tr>
<tr>
<td>$\eta = 0.25$</td>
<td>0.874</td>
<td>0.857</td>
<td>0.851</td>
<td>0.844</td>
</tr>
<tr>
<td>$\eta = 0.10$</td>
<td>0.743</td>
<td>0.894</td>
<td>0.696</td>
<td>0.888</td>
</tr>
</tbody>
</table>

8 Conclusion

This study shows an artificial neural networks used in splice site prediction. The best neural network trained in this study, achieve a correlation coefficient at 0.568. This result is achieved without any prior knowledge of any sensor signals, like 'GT' or 'GC' for the donor splice sites, or 'AG' for the acceptor splice sites. Also note that some of the genes in the data sets did not store the base case for splicing, but an alternative splicing, which may have disturbed some of the training. It is fair to conclude that artificial neural networks are usable in gene prediction, and the method used, with a sliding window over the gene, is worth further study. This method combined with other statistical methods, like General Hidden Markov Models, would probably improve the results further.

References
Bibliography

[1] BLAST, Basic Local Alignment Search Tool. NCBI Web-site.


