Learning Representations of Sequences using convolutional restricted Boltzmann Machines

Roman Schulte-Sasse
r.schulte-sasse@fu-berlin.de

Masterthesis
Freie Univesität Berlin
Fachbereich Mathematik und Informatik

Supervisors
Prof. Dr. Annalisa Marsico
Prof. Dr. Martin Vingron

Berlin, July 27, 2016

ABSTRACT

Deep learning for biological problems has become an active topic of research quite recently. While deep learning techniques have already been shown to be very effective in image recognition, their application to complex biological problems was postponed for two reasons. It is only until recently that data on such a huge scale as needed for deep architectures can be provided in the field. And furthermore, the interpretability of neural networks has been an issue in the past.
The latter problem is overcome by convolutional neural networks while the rise of next-generation sequencing (NGS) generated the large amount of data required for training deep architectures.

Such methods learn representations of the data they are confronted with during training. Complex representations of genomic sequences are greatly needed to further understand the regulatory mechanisms of cells on a molecular level as these are even more complex.
Restricted Boltzmann Machines (RBMs) are a method to learn such data representations in an unsupervised fashion, that is without requiring a label for each training point. This property makes RBMs an ideal tool to work on NGS data and extract features from that data.
To solve the issue of interpretability, a convolutional variant of RBMs, similar to convolutional neural networks has been shown to learn meaningful and interpretable features from data in the field of computer vision.

In this thesis, I use convolutional restricted Boltzmann Machines (cRBMs) to learn from NGS data. I show that the extracted features are meaningful and can easily be interpreted in a biological setting. Furthermore, these features are specific to the set of sequences that the model has been trained on, allowing for classification of tissue types.
Since the method works unsupervised, it can be used to detect biases in NGS protocols, such as PAR-CLIP which measures interactions between RNA binding proteins and their corresponding RNAs.

DECLARATION OF ACADEMIC HONESTY

I hereby declare that this master's thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given in the bibliography.

Roman Schulte-Sasse Berlin, July 27, 2016

ACKNOWLEDGEMENTS

I would like to thank Prof. Dr. Annalisa Marsico for her support and guidance throughout the whole thesis. Thanks also to Wolfgang Kopp for his constant feedback and very productive collaboration.

My thanks go also to Marie for the fruitful discussions and her productive support throughout the last years. Thanks to my flatmates for proof-reading and hanging around on the couch.

Last but not least, thanks to everyone in the RNA bioinformatics group for helping me find an entry point to bioinformatics and biology.

CONTENTS

1

INTRODUCTION

Since the rise of DNA-sequencing in the early 2000's, there has been a dramatic increase of available biological data. New methods to analyze and classify RNA and DNA sequencing data were developed and have constantly been improved since then.
Knowledge of the structure and patterns in genomic data is essential for many tasks, following the hypothesis that sequences sharing a common pattern do so for functional purposes. This hypothesis has been confirmed in several experiments and is therefore taken as a common assumption for many algorithmic approaches in the field of sequence analysis [14] and makes it possible to transfer algorithms originally designed for pattern recognition to the area of sequence analysis.
While there are many tools and algorithms around to perform specific tasks, such as *motif discovery* or *peak calling*, there is a lack of data representations that are not bound to a problem but rather describe genomic sequences independently from it.

Deep learning methods have gained much interest in the last years. It has been shown that deep neural networks are the state-of-the-art technique to interpret images and videos.[30, 51, 48] They are promising in decision making when no analytic solutions can be found and have even led people to claim that the need for programmers has come to an end. While the latter is not very probable for the time to come, deep architectures have had great success in many applications.
A significant part of that success comes from so-called *convolutional neural networks* because they exploit some properties that images have in common. Genetic sequences share many of these properties as for instance the importance of context for a given pixel or nucleotide for its function.
Thanks to the aforementioned boom in available genomic data it has become possible to apply these algorithms to biological problems, as shown in [2, 61, 26].
However, all of these algorithms only work in a supervised setting

in which labeled data is available. But the generation of labels for sequencing data (such as *RNA expression values*) is often expensive.

In this thesis, we apply the *convolutional Restricted Boltzmann Machine* (cRBM) to genomic sequences. CRBMs are an algorithm to learn a representation of data in an unsupervised fashion, i.e. when labels are not available. The algorithm is very similar to a convolutional neural network and has been shown to find good data representations in computer vision applications [32].
We show that the learned model is structured in a way to solve the task of motif discovery naturally when learning to represent the data well.
Furthermore, we use trained models to classify open chromatin regions into tissue-specific regions. This is done by comparing the likelihood that a sequence belongs to either of the two data distributions learned previously by the cRBM.
And finally, we use that same likelihood to detect biases in high-throughput sequencing protocols such as the one used to generate RNA sequences which bind a protein of interest, called PAR-CLIP.

1.1 CHALLENGES IN SEQUENCE MODELING AND SCOPE OF THE THESIS

As mentioned already, we test our model on three common problems in sequence analysis and evaluate it. While the cRBM is not specifically designed to solve any of them, there have been applications in other domains. For instance, RBMs were used to learn meaningful features in images [32] and to predict if events belong to one or another class of events [27].
While the cRBM model has been applied in different fields of science, [32, 40, 24] there has not yet been any approach on genomic sequences to the best of the author's knowledge.

1.1.1 *Motif Discovery*

Short reoccurring patterns in genomic sequences can be interpreted as *sequence motifs*. Such a motif describes a pattern of nucleotides to which a certain protein will bind much more likely than to other regions of the sequence. These patterns are typically only preferences for the protein and are therefore described in a probabilistic way. The most common way of describing the binding preferences of RNA/DNA-binding proteins is by using *position weight matrices* which are matrices of dimensions $k \times |\Sigma|$ where k is the number of

nucleotides of the preference and Σ is the alphabet of possible nucleotides.
While there are known experimental setups to detect motifs in sequences, most of the work is done by first sequencing cells with next-generation-sequencing techniques, followed by the application of algorithms to analyze these often massive amounts of data.
Traditionally, algorithms that perform discovery of *de novo* motifs do so by modeling patterns in the data in a probabilistic way. They determine if such a pattern is over-represented in the data. The problem of motif discovery is often solved with machine learning techniques.

In this thesis, I will present the convolutional restricted Boltzmann machine as a method to find a representation of sequences as a linear combination of motifs. I will show that motifs correspond to *kernels* in convolutional architectures for computer vision tasks and that algorithms exist to learn them.

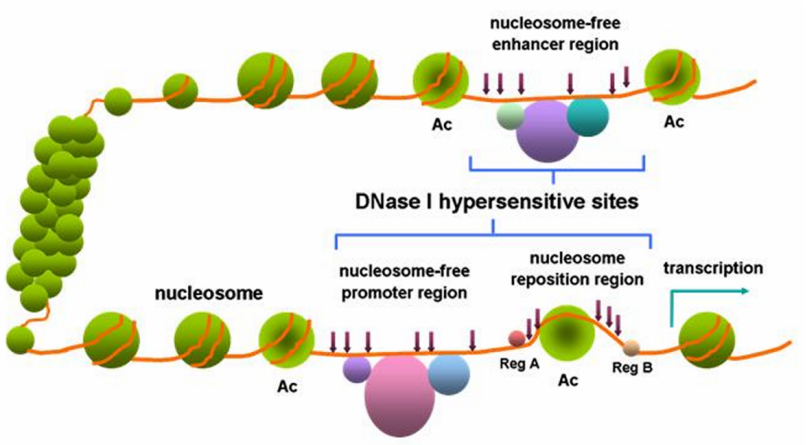

Figure 1: The open chromatin regions.
Source: [55]

That way, the cRBM finds *de novo* motifs when being trained on a set of similar sequences.
We show that the cRBM can recover transcription factor binding sites which are known from literature on two different data sets in a DNA context when learning to represent the data well. For this task, the training data are relatively short sequences from *DNase I hypersensitivity sites* (DHS). DHSs are regions in the chromatin which are sensitive to the DNase I enzyme. The DNase I enzyme cleaves the DNA in these regions, thus giving them their name. A schematic view of DHSs is depicted in figure 1.
Since their discovery some 30 years ago [59], DHSs have been used as markers of regulatory DNA regions as these areas are mostly free of nucleosomes. [52] DHSs are associated with different regulatory

elements, such as promoters, enhancers, and silencers to just mention but a few.[19]
The training data are thus sequences which are expected to have strong patterns in them which can be identified.

1.1.2 *Classification of Tissue Types*

The learning procedure solves both of the problems in motif discovery while also forming a probability distribution over the sequences. We use that likelihood probability for our sequences by evaluating it for two different models and choosing the class with higher probability. That way, it is possible to determine whether a novel sequence comes from the tissue type of sequences the model was trained on, or not.
Using test sets from DHS regions, we show that this classification yields accurate results which are comparable with state-of-the-art classifiers. Good classification results are an indicator that the features learned by the cRBM are not only meaningful in general but are also distinct over different training sets.

1.1.3 *Bias Detection in Sequencing Methods*

The likelihood of a sequence given the cRBM model states a probability that it belongs to the data on which the model was trained on. We build a model upon sequences which come from PAR-CLIP control experiments. PAR-CLIP experiments measure interactions between RNA sequences and a protein of interest.

In a PAR-CLIP experiment, all bonds between RNA and proteins are fixed and then a specifically designed antibody "drags out" the proteins of interest together with the RNAs. The results are then reads which in theory correspond to RNA binding protein binding sites.
There are several different occasions where a bias might occur in that kind of experiment because the antibody does not only pull out the proteins (and RNAs) of interest but also others [17].
While there have been attempts to tackle the problem of background binding for different protocols, there is still no method to measure background binding accurately.

In our experiment, we focus on the PAR-CLIP protocol. While it is known that structured biases exist for various biological reasons[17], there have been only a handful of approaches to analyzing that bias accurately and normalize for it.

1.2 OUTLINE

In this thesis, I will first present the current state-of-the-art in all of the problems mentioned above in chapter 2. I will then give a detailed mathematical derivation of the restricted Boltzmann machine and its convolutional counterpart in chapter 3. There, I will describe the modifications of the original algorithms that were used to enforce certain properties of the model.
In chapter 4, I will present experiments for all three of the problems and show that the cRBM model can recover motifs of known transcription factor binding sites in DNase I hypersensitivity sites for different tissues. I will further show that a classification into various tissue types can be done with our model and that the cRBM compares well with sophisticated supervised classifiers.
Finally, I will present the results of the analysis of PAR-CLIP control experiments, leaving the domain of DNA sequences and moving to the more complex field of RNAs. While this is promising, there are still many improvements which are yet to be implemented. I will lay them out in chapter 5 of the thesis.

2

RELATED WORK

There is a variety of algorithms that try to solve the challenges mentioned in the previous chapter. The extraction of *de novo* motifs from genomic sequences (motif discovery) is typically done by generative models, such as *hidden markov models* (HMMs) [35] or *expectation-maximization* (EM) [4]. Other approaches use the notion of *k-mers* to classify sequences into protein families or cluster transcription factor binding sites.
K-mers are all possible substrings of a given length k of a string, given an alphabet Σ. In a DNA context, Σ typically contains the four letters $\{A, C, G, T\}$, representing the four nucleotides of the DNA. They can be used to represent a sequence as a vector, giving rise to different string kernels. These are similarity measures between two strings based on the dot product of two such k-mer vectors.
Various machine learning techniques have been proposed for protein classification in the past, ranging from relatively simple linear models to complex systems like *support vector machines* or neural networks. String kernels have made it possible to classify sequences with SVMs (SVMs are further discussed in chapter 2.1.4) and they have reached very high accuracy in such tasks.
However, with the availability of big data in bioinformatics, more complex methods can be applied to the problem of classifying proteins into different families or classifying transcription factor binding sites according to their function.

The very same applies for the problem of motif discovery. While the problem itself is NP-complete [42], many approximation algorithm exist, the most important of them being *MEME* which is based on the *expectation maximization algorithm* (EM) [4].
With the large amount of data available, these methods are limited in terms of computational time required for training. Newer variants of MEME designed for NGS-data only process a subset of the data, neglecting the rest [36].

In the following chapter, I want to give a brief introduction into known approaches to motif detection and classification, especially

discussing the EM-algorithm used by *MEME*. Previous work in the domain of pattern recognition in images have focused on neural networks and we will examine the foundations of that technique in more detail. Since artificial neural networks lay out the foundations of the restricted Boltzmann Machine and because they have been applied so successfully, we will review their training algorithm more closely. From there, I will present the convolutional neural network (cNN) as an extension to the original algorithm and then give a general introduction on restricted Boltzmann Machines. Finally, some very recent applications of deep learning in biological problems are introduced, especially the *deepBind* network by Alipanahi et al. to predict effects of *single nucleotide variants* from sequences alone.[2]

2.1 APPROACHES TO SEQUENCE MODELING

In this section, we want to give an overview of existing methods for motif discovery and modeling of DNA. The task of discovering *de novo* motifs in DNA sequences can be regarded the same as finding patterns in text [14].
However, the problem has to be solved probabilistically because binding proteins seem to favor some regions and neglect others but are not deterministically binding to a specific pattern while never binding to another. This is why sequence specificities of proteins are usually described using position weight matrices (PWMs) [34]. MEME tries to find de novo motifs in sequences by fitting a mixture model to the sequences, using an expectation maximization approach.[4] The algorithm is explained in more detail in section 2.1.1. MEME finds one motif at a time in the sequences, interpreting the whole task as an unsupervised machine learning problem [4].

Other tools work similarly, and most algorithms model a probability to distinguish between "interesting" regions (motifs) and some background, [34] exploiting the locality of transcription factor binding sites. Motifs are found by looking for patterns which are overrepresented in the dataset. Such a motif occurs more often in the data than would be expected if it was generated by chance. Such a frequentist modeling of sequences also involves the notion of background in order to determine the frequencies for each of the letters in the alphabet (A, C, G, T in DNA).
However, more recent approaches show that modeling sequences as PWMs might not be enough to capture the whole biological meaning encoded in them. It is partly because of the more sophisticated mathematical representation as an ensemble of PWMs and their interactions on a higher level that *deepBind* is outperforming other tools

in many aspects [2].

2.1.1 *MEME and the EM-Algorithm*

As already mentioned in the previous paragraph, *MEME* is the most prominent tool to detect new motifs in a given set of sequences [34]. It tries to maximize the probability to observe the data in an iterative fashion, assuming a particular model of observed and unobserved variables. That can be interpreted as searching for the model that most likely has generated the observed data. These models are therefore often called *generative models*.
In MEME, a latent random variable (RV) (one that can not be observed but is responsible for certain properties of the observation) describes whether a sub-sequence of a fixed width belongs to the motif or rather to the "background". The background is modeled as a vector $\Theta_2 \in \mathbb{R}^{|\Sigma|}$, containing the frequencies of each letter in the alphabet. The motif, on the other hand, is modeled as a PWM with k columns and $|\Sigma|$ rows.
The sequence $x_i \in \mathbb{D}$ is partitioned into sub-sequences of length k with overlapping windows, meaning that frame of length k is "slid" over the sequence. It advances by one nucleotide with each time step. This windowing approach is similar to the cross-correlation operation which is explained in more detail in chapter 3.3.1.
Furthermore, a mixing component λ states the probability of a window to belong to either the background or the motif.

The EM-algorithm uses the concept of a *hidden variable* that was important for generating the data but can not be observed. It iteratively estimates first these hidden components, assuming that the other parameters are all known and then estimates the parameters, assuming that the hidden components are known.

A much simpler approach than the EM-algorithm is the use of a *first-order hidden markov model*. While also being generative, such a model assumes independence between most of the RVs.

2.1.2 *First Order Markov Models*

First order markov models try to model sequences using a *markov chain*. A markov chain consists of random variables at a certain point t. By counting how often transitions from one nucleotide (modeled as RV here) to another occur in the training data, transition proba-

bilities can be estimated. Markov chains further fulfill the markov assumption, stating that the transition probabilities from one letter to the next only depend on the previous state, the chain was in but none of the states before that one. Visualizing this process as graphical model looks similar to a chain (compare to figure 2), hence the name.

Figure 2: Visualization of a markov chain as graphical model. Transition from one state to the next is always independent from previous states which simplifies the possible transitions.

To determine if a motif is over-represented in the data, a background probability for each letter is typically calculated as well. For new sequences, we can sum over all logarithmic transition probabilities in a sequence in order to calculate the log-likelihood of the sequence, given the model.

$$\mathcal{LL} = \log \prod_{i=1}^{N} Pr(x^{(t)}|x^{(t-1)}) = \sum_{i=1}^{N} \log Pr(x^{(t)}|x^{(t-1)})$$

First-order markov models are appealing due to their simplicity but they lack contextualization of nucleotides into their surroundings. Furthermore, they do not assume that there are unobserved components in the generative process.
This is why we use them in chapter 4.4 as a lower bound for classification.
A model that overcomes the weaknesses first-order markov models is the *hidden markov model* (HMM) which introduces the notion of hidden variables and models them while still guarding the markov property.

2.1.3 *Hidden Markov Models*

Hidden markov models are an extension of the classic markov chain. Much like the EM-algorithm, HMMs try to model underlying hidden variables that are important for the generation of data but cannot be observed directly. The state of these latent variables has to be inferred instead.
The goal in a hidden markov model is to maximize the likelihood of the data. This maximization sounds familiar and indeed, HMMs

can be trained with the EM-algorithm. Markov models, HMMs and also the restricted Boltzmann Machine introduced in 2.3 belong to the class of generative models.
Training HMMs is usually done with the *Baum-Welch algorithm* which is a specialized version of the EM-algorithm. It iteratively optimizes the parameters of the model and infers the hidden states with it.

So far, I introduced several generative models for the problem of sequence modeling. Classifiers, however, also often have to convert genomic sequences into mathematical notation prior to classification. One such an approach are *string kernels* which are extensively used by *support vector machines* (SVMs).

2.1.4 *Support Vector Machines*

A support vector machine is a maximum margin classifier. It follows the theory that a larger margin between the closest point from a class and the separation line leads to a better generalization of the model. For that, SVMs consider both, positive and negative data for the selection of an optimal decision boundary between the two classes.
SVMs are linear classifiers but can easily be *kernelized*. Kernelization means transforming the input data to a usually much higher dimensional space. In such a high-dimensional space, classification problems often become linear, even when they are not in the input space. Because SVMs only need to have a similarity measure between two data points but do not need the high dimensional space at any other point, efficient mathematical approaches exist to transform the data to the kernel space. Such approaches make sure that the data is never actually converted to the high dimensional space explicitly but only implicitly.
Kernelized versions of SVMs can solve non-linear problems while still finding an optimal solution for the problem in a given kernel space. However, one must select a kernel explicitly, thus introducing expert knowledge and a potential bias. Furthermore, SVMs are not easily interpretable. It is a hard problem, for instance, to select the most important features for the formation of the decision boundary which often can only be solved by trying out all different combination of features and looking at the loss.
SVMs find sparse solutions for a given classification by selecting so-called *support vectors* which contribute to the formation of the decision boundary. All other points can be discarded after training which is a considerable advantage over most other classifiers (like k-nearest neighbors) when dealing with large data sets.

2.2 NEURAL NETWORKS

During the last years, the focus has shifted from biological experiments to verify functions of genes or transcription factors to more computational approaches (for example in *gene finding*). Machine learning has become increasingly important in many areas of bioinformatics and often generates new insights to the mechanisms of gene regulation.
Two different approaches can be seen in the machine learning community, one being the range of classifiers that try to discriminate data into different classes. They can be very powerful but rely on the labels (i.e. they only work supervised).
Another approach is the generative models, such as markov models, HMMs or RBMs.
While motif discovery has traditionally been solved with generative models, recent advances have used a classifier for this task that was detecting motifs without explicitly looking for them [2].
On the other hand, protein classification and gene finding were usually solved with support vector machines, logistic regressions or neural networks.

The latter has evolved to be one of the most influential classifiers during the last years. Today, artificial neural networks are the state-of-the-art machine learning system for applications in computer vision, natural language processing and robotics.[62, 10, 60] and have come a long way since the first description of the perceptron algorithm in 1958.[45] The perceptron was a simple linear classifier that could be trained by showing it example data vectors and yielded an optimal solution if there was one available (that is, the data was linearly separable).
However, this first algorithm was not embedded in a network of other classifiers, and it was unable to solve any non-linear problems and especially not the famous *XOR-problem*. This most simple 2-dimensional problem depicted in figure 3 led to a serious decline in research activity in the field. [47]
It was only until 1974 that Paul Werbos created the *back-propagation algorithm* that was the first training algorithm for networks of different percep-

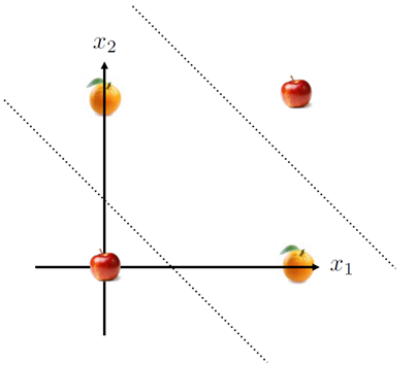

Figure 3: The XOR problem with apples and oranges

trons.[56] That algorithm lead
to an enormous re-interest in the field and formed the basis of modern artificial neural networks (aNNs). But even with an effective learning algorithm at hand, aNNs with many hidden layers remained un-trainable because of the nature of the backpropagation algorithm. For a training example, an error is calculated that typically lays between zero and one. When this value is now propagated through a deep aNN, this error becomes smaller and smaller, leading to the *vanishing gradient problem*. The lowest layers of the aNN are not trained any longer by backpropagation, leaving these most important layers with their random initialization.
However, recent advances have overcome the problem by pre-training the aNN in a layer-wise fashion or by introducing new kinds of non-linear activation functions. Together with recent advances in hardware and especially GPUs, the path was clear for what today is dubbed *deep learning*.
Deep learning architectures are aNNs that have multiple hidden layers and are so very powerful because the lower layers learn useful data representations that can be used by higher neurons in the network to perform classification. Until now, there exists only a handful of applications of deep learning to biological problems. Since it is usually hard to get large data sets in life sciences and since aNNs typically need very high amounts of training data, that fact is entirely understandable. But with the rise of *next generation sequencing* over the last years, it has become possible in genomics to generate large amounts of sequencing data that can be used by deep learning applications.

In a more formal way, we can define a fully connected feed forward neural network as a directed and acyclic graph in which the nodes are arranged in layers. Nodes within one layer have no connections to each other but every node from layer i is connected to all nodes in layer $i+1$ and $i-1$.
The connections between nodes have weights associated with them. Nodes can also be called neurons, or units. To classify a data vector, it is clamped to the lowest layer of the network. From there, the numbers are propagated to the next layer and each neuron sums together the incoming numbers and decides whether to activate, depending on a non-linear function, typically the sigmoid function.
This way, each neuron can be viewed as a linear classifier that takes a certain responsibility in the search space. In image recognition, a neuron might be responsible for detecting horizontal edges in an image, while a neuron further up the network might be responsible for deciding whether there is a cat in the image or not.
Feed forward neural networks are a supervised method, which means they need labels for input vectors in order to learn from the data. When a new training vector is shown to the network, the informa-

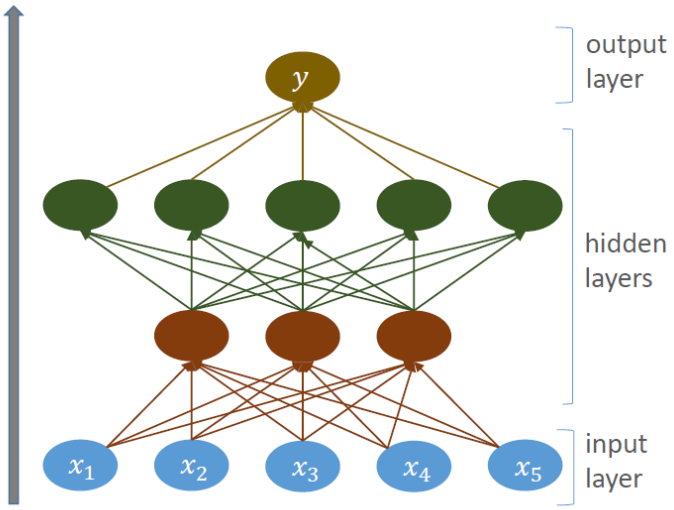

Figure 4: A schematic view of a fully connected feed forward network. The 5-dimensional input vector is first forced to be represented by only three dimensions (red layer). This reduced representation is then seen by the next layer (green layer) which tries yet again to find another representation of the input data.Finally, a one-dimensional output classifies the input vector.

tion is first propagated up. At the output layer, it is compared to the desired output (label) and an error is calculated from the difference of both. This error is then used to calculate derivatives at each of the edges. The weights are then changed a little into the direction of that gradient, and the procedure is repeated. This gradient descent algorithm is a very popular choice in machine learning and can be applied when there is no analytic solution to the maximization of the objective function. Unfortunately, the algorithm does not give any guarantees as to convergence rates or global goodness of the final solution.

Calculating the partial derivative of the error function with respect to one of the weights is an iterative process. First, the derivatives for the deepest layer are calculated and from there it becomes possible to calculate them for the second-deepest one and so one. This top-down approach is why the algorithm is called backpropagation, because it starts at the back and propagates the partial derivatives back to the front.
In the next section, we want to dive more into the training procedure of feed forward aNNs to see in how far it is similar to our proposed method and where it differs.

2.2.1 *Training Neural Networks*

The backpropagation algorithm tries to answer the question, in how far each of the neurons are responsible for the error that was made by the network as a whole. That means, when we propagate the error up to the output layer, we then first calculate the output's error, using the *squared error* measure.

$$E = \frac{1}{2}\sum_{i=1}^{k}(y_i - t_i)^2$$

This error is then split up between all neurons in a top-down fashion. The key principle is here to calculate the error's partial derivatives with respect to the weights that connect that two neurons. We start by calculating the derivative for the weights connecting the output layer with the second-last layer. These can be calculated easily with the *chain rule*. From that point, it becomes possible to calculate the derivatives for the weights connecting the second-last with the third-last layer and so on. Finally, we arrive at the input layer and the backpropagation algorithm terminates for the given input. When we repeat this procedure for all data points, the network will slowly start to learn to minimize the error.
We will assume that the neural network at hand uses the *sigmoid activation function* which is given by:

$$sig(x) = \frac{1}{1 + e^{-x}}$$

This function is particularly handy because of its very easy derivative. We can write:

$$sig'(x) = sig(x)(1 - sig(x))$$

So the question that asks in how far one unit is responsible for the error of the whole network becomes a question of partial derivatives with respect to the connections between layers (the weights). The *chain rule* states how the derivative for functions that were applied one after the other can be written, using only the single functions. More precisely, the chain rule states:

$$(f \circ g)' = (f' \circ g) \cdot g' \tag{1}$$

When we consider the neurons as functions that are applied one after the other, we can finally calculate the partial derivatives.
During the forward step, the output and input of each neuron is stored. Let o_i^k be the output of neuron i in layer k and h_j^{k+1} be the input for neuron j in layer $(k+1)$. These values are known from the forward pass and can be used to calculate the partial derivative with respect to the weights.

$$\frac{\partial E}{\partial w_{ij}^k} = o_i^k \delta_j^{(k+1)}$$

where δ_j^k is given by:

$$\delta_j^k = \begin{cases} f'(h_j^k)(t_j - o_j^k), & \text{for output layer.} \\ f'(h_j^k)(\sum_{i \in incoming} w_{ji}\delta_i^{(k+1)}, & \text{for inner layers.} \end{cases}$$

There are several interesting properties in the backpropagation algo-

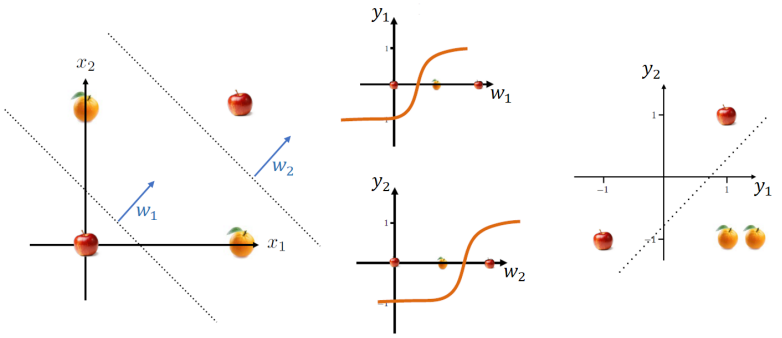

Figure 5: Example of how a neural network can solve the XOR problem. The first layer of the network only finds good data representations that serve as input for the next layer. There, the problem is suddenly linearly separable, and the problem as a whole can be solved.

rithm that do not seem evident from the formulas. First of all, the network tends to find useful representations of the data in the first layers, as depicted in figure 5. Then the choice of the non-linear activation function plays a fundamental role as to how the network is able to learn. But even with a non-linearity at each neuron, a feed forward network can be regarded as ensemble of different linear classifiers, each of them being responsible for one part of the overall task. Another observation is that the backpropagation algorithm is highly efficient because of the reuse of values that were computed for the forward pass already. When we use the sigmoid activation function with its easy derivative, we can write the entire partial derivative as a combination of o_i^k and $\delta_j^{(k+1)}$.
Despite all of the advantages of neural networks to other classifiers, they have also some disadvantages. Because of the large amounts of parameters, the classical fully connected feed forward network requires huge amounts of training data for successful learning. Overfitting can occur easily and does so frequently. Furthermore, there is no guarantee that the solution found by the network is optimal in any way. And finally, applying these kinds of networks to images is usually not feasible due to the huge amount of parameters and the very structure of an image. When there is the same pattern of pixels somewhere in the image, we usually want it to be recognized as such, no matter where the pattern is located in the image. But imagine a neural network that is trained to see some feature (an apple, for

instance) in the upper left corner. The network would be unable to recognize the very same apple located in the lower right corner of the image when this image was not part of the training set.

To overcome these problems, LeCun et al. proposed the convolutional neural network in [31], which resolves most of the problems in the field of computer vision applications.

2.2.2 *Convolutional Neural Networks*

The convolutional neural network is a variant of the conventional neural network in which the connections between two units are convolutions instead of simple connections. The convolution operation is explained in detail in section 3.3.1 but it is worth noting that convolutions can be regarded as filters or kernels applied to images.
With this informal definition, we can define the weights connecting the input layer with the first one as filters. Their application to the image (the input layer) yields in the first hidden layer of the network. When we have a two-dimensional image of $x \times y$ input neurons, the first hidden layer typically is three-dimensional. This is because we apply many filters to the image, each one of them producing a two-dimensional matrix.
When we now look at the mathematical side of it, we only have to change the way gradients are computed from the error. Since the connection between layers is a convolution, we can also express the gradient as convolutions.

This approach solves the translation invariance issue because a filter will be applied everywhere in the image. Furthermore, the number of parameters is drastically reduced. When we use k different kernels, each of size $x_k \times y_k$, we have $k \cdot (x_k \cdot y_k)$ parameters for the convolutional layer. Typically, x_k and y_k are small and 20-30 kernels are often enough to describe the structure of an image.

For image -and biological applications, the interpretability is increased when using convolutional neural networks because the kernels have an interpretation that is known already. In image processing application, the kernels usually correspond to stereotypical mini-images like the cat or the face in figure 6. In biological applications, however, the kernels can correspond to *position weight matrices* in the first layer. These matrices can be visualized as web-logos and then interpreted in a very humanly-readable way.
Later stages could be visualized either as combinations of motifs in the input space (that is as a web-logo, scaled by the amount of lower-level PWMs contributing to the feature). Or they could be visual-

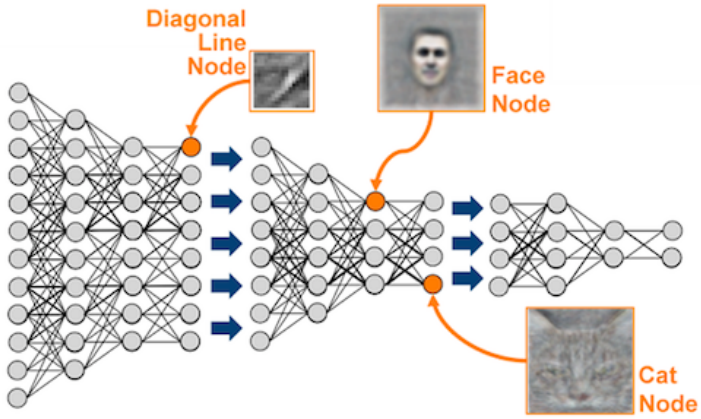

Figure 6: Schematic view of a deep convolutional neural network. The first layers capture low-level features like edge-detectors in the images. Later stages of the network capture more complex features like cats or faces. Source: [39]

ized as interactions between transcription factors in an interaction network.

We can see that by exploiting some prior knowledge about the domain from which the data originates, it is possible to solve many problems that arise with neural networks. Convolutions are very common in computer vision tasks but also in motif discovery. When we want to find a motif in a DNA or RNA sequence, this can be done efficiently with convolutions.

2.3 RESTRICTED BOLTZMANN MACHINES

We have seen that cNNs are an adaption of the original aNN and that this extension to a convolutional context can be embedded easily into the backpropagation algorithm.
The restricted Boltzmann Machine can be seen as an unsupervised alternative to neural networks. Much like them, RBMs learn useful data representations but unlike aNNs, the usefulness is not defined as being good for classification. Instead, RBMs belong to the class of generative models. Hence, the model tries to maximize the probability that its parameters have generated the training set. With that approach, it captures abstract structures in the data.

A restricted Boltzmann Machine is a two-layered undirected graphical model with no connection within one layer. Each node represents a random variable (*RV*) and edges have weights assigned to them. An edge represents the statistical associations between nodes (RVs). The

units are *fully connected*, that is every unit in one layer is connected to each unit from the other layer.
In graphical models, the random variables can be either observed or not. When they are observed, the current value of the RV is known. Following the rules for statistical independence, we can see already that an observed variable "blocks" the dependence of RVs.
Because of the lack of connections within one layer, each of the units become independent from each other when the other layer is observed. This property is critical to formulating efficient algorithms for this kind of graphical models and is the difference between "standard" Boltzmann Machines and restricted Boltzmann Machines, so called *RBMs*. Restricted Boltzmann Machines, as any graphical model,

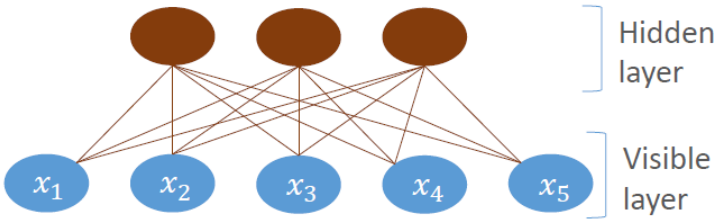

Figure 7: Schematic view of a RBM with five input units (blue) and three hidden units (red). Units in the same layer have no connections while every unit from one layer is connected to each of the units from the other layer.

build a probability distribution $P(x)$ over the data. A RBM models the data by introducing a hidden layer that describes structures or categories of the data.
A random variable in the hidden layer stands for one of these categories within the data and when we have a situation in which the number of hidden units is smaller than the number of visible ones, RBMs find a representation of the data in a lower-dimensional subspace.[24]
The goal of training a RBM is to learn these categories from the data without the use of labels, such that the learning can be interpreted as solving a clustering problem.

A specialty of the RBM is that all units (nodes) are binary, that is they can have states of either zero or one. That indicates whether a particular category is activated or not in H or V. As we will see in chapter 3.3.4, DNA sequences can be converted to a matrix in which the binary requirement is met.
A restricted Boltzmann Machine works similar to a neural network in that the data is clamped to the model (to the visible layer) and then propagated upwards first. By clamping the data vector to V, the random variables in V become observed. By the rules of graph separation, H then becomes statistically independent from V. When we want to propagate the information back down, the statistical indepen-

dence holds for the units in the visible layer because the whole layer H is observed.

We have mentioned before that the hidden layer performs clustering once the model is properly trained. Input vectors will switch on some of the units in the hidden layer. In that way, H is a representation of the data and can be used to reconstruct it. It has been shown that RBMs can restore information when only partly completed data vectors are clamped to V [49].

After introducing the traditional generative methods for modeling sequences, we saw how deep neural networks could learn representations on their own for classification. RBMs are a hybrid in the sense that they, as neural networks, learn data representations automatically. However, they belong to the class of generative models.
Recent advances in bioinformatics have made use of deep neural networks to predict binding affinities *in silico* and their model learns motifs as a byproduct. This application shows that deep learning techniques have very promising use cases in bioinformatics, and I will present their approach in more detail.

2.4 PREDICTION OF BINDING AFFINITY WITH CNNS

The success of aNNs with multiple hidden layers comes mainly from the extended possibility for the network to find a good representation of the data. Due to the hierarchical order of different layers, the classification problem in a deep neural network is split into several pieces. Each of the layers can be seen as just augmenting the data representation a little by either adding or removing information from it.
Smaller hidden layers create a bottleneck of the information flow, forcing the network to learn a sparse representation of the features from the previous layer. Such a compression can be very useful for classification because unimportant parts of the data are neglected, and noise is removed.
But also the translation equivariance of convolutional approaches contributed to the success story of deep learning techniques. These networks require considerably less parameters for training and boost the interpretability of the model at the same time when looking at images or genetic sequences.

In 2015, the paper by Alipanahi et al. received huge attention.[28] The group predicted the effects of single nucleotide variants (SNVs) on a molecular level and outperformed all of the state-of-the-art methods by using a deep convolutional neural network for the prediction.[2]

Not only was their tool, called "DeepBind", able to accurately discover motifs in sequences, it also yielded top results in determining the effect of SNVs from sequence alone.
As shown in section 2.2, deep neural networks learn representations of the data that is presented to them during training. And since convolutions of sequences with PWMs is equivalent to scanning sequences for motifs as mentioned in the previous section, the first layer of a cNN learns motifs and thus transcription factor binding sites. In the domain of motif discovery, deepBind outperformed all exist-

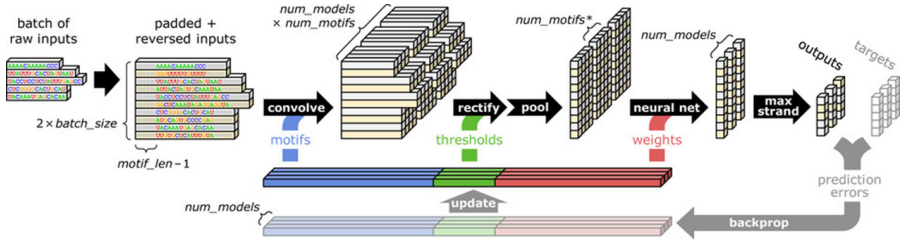

Figure 8: The architecture of the deepBind model. The convolution layer is followed by max pooling and a densely connected layer with one output neuron. Source: [2]

ing methods in the *DREAM5 TF-DNA Motif Recognition Challenge* in which a set of sequences from *protein binding microarray* (PBM) experiments is used. While all of the sequences are made known, only a some of the PBM data was provided. The goal of the challenge was thus to predict the probe intensities of the array.
To see whether algorithms perform well across sequencing technologies, the same trained model was also evaluated on 500 ChIP-seq peaks. The motifs, however, are just one part of the learned representation. The dense layer that is applied after the convolutional layer also has a biological meaning. It learns certain patterns in the layer of "motif hits". This way, the authors of the deepBind paper were able to infer rules for combining these patterns into one score that predicts binding.
To analyze the effects of SNVs, the authors compared the predicted score for the original sequence and for a modified sequence with one nucleotide changed. When the score changes significantly, then the SNV is likely to either damage the gene function or adding new functionality.

The advantages of a deep learning approach to the problem are the ability of neural networks to deal with large amounts of data in a reasonable amount of time and the hierarchical order of the layers. The learned data representation solves the motif discovery problem as a byproduct of maximum classification and also finds patterns in the co-activation of motifs.

2.5 APPROACHES TO FIND BIASES IN PAR-CLIP RNA CONTROL EXPERIMENTS

Friedersdorf and Keene showed in [17] that background binding introduces a significant bias to high-throughput sequencing methods such as PAR-CLIP, which profiles RNA binding protein sites.
In their study from 2014, the authors conduct and analyze control PAR-CLIP experiments, revealing that background binding is common. A control experiment is one where instead of pulling RNA from the cell with an antibody that only binds to the protein of interest, an unspecific antibody is used. In theory, such an experiment should result in a uniform distribution of RNAs throughout the cell. However, this is not the case as Friedersdorf et al. have shown. [17]
Instead, there are motifs also in the data from control experiments. Furthermore, the authors claim that a profound analysis of the background binding could significantly improve existing methods to call peaks in such data sets. By incorporating information from control experiments into their analysis, the authors were able to show that new binding sites could be identified for an RNA-binding protein (RBP) with unknown binding preference.
In their work from 2014, the authors did not develop or evaluate a method for peak calling that takes into account the biases from control experiments. A newly developed method would have to deal with large amounts of data and should be able to work in an unsupervised fashion.

There are different peak-callers for various protocols for high-throughput RNA sequencing methods available. They typically model the read-count distribution of a data set using variants of the negative binomial distribution [54, 43] and assume that this also models the background noise distribution. A peak is then found by looking at sites where the read-counts are significantly larger than expected. However, the above-mentioned approaches do not model the background explicitly from control experiments.
Detection of such biases in an unsupervised fashion can help the peak calling process and interpretation of the final results.

3

STATISTICAL METHODS

In chapter 2 we saw an intuition on what restricted Boltzmann Machines (RBMs) are learning and what they can be used for. In this section, I will re-introduce RBMs in a more formal way, show the derivations of the most important formulas and describe how we applied convolutional RBMs to biological data.
For that, I start by introducing some mathematical concepts on which RBMs are relying. From there, I present RBMs as probabilistic graphical models and introduce its convolutional counterpart, the convolutional RBM. This model is an extension of the original algorithm and can be regarded as a convolutional neural network with stochastic activation.
Finally, I will introduce some methods that we used to enforce sparsity on the model and to avoid the problem of overfitting.

3.1 MATHEMATICAL FOUNDATIONS

We will first discuss *gibbs sampling* and *block gibbs sampling* as the most important algorithms to obtain samples from a complex distribution where this is not possible otherwise. I will then present the *gradient descent algorithm* as the method of choice to do learning in a non-convex optimization environment and finally introduce the softmax and multinomial distributions as they are crucial to sample from multiclass probabilities. This is often the case in our application on sequences to sample a nucleotide from a distribution over all nucleotides.

3.1.1 *Gibbs Sampling & Blocking Gibbs Sampling*

Gibbs sampling is an algorithm to draw samples from a multivariate probability distribution of which it is complicated to get samples of.

It is used when we want to draw samples from the joint probability distribution $Pr(x_1, \ldots, x_K)$ but when we can only get samples from the conditional distributions $Pr(x_i|x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_K)$.
To get a true sample from the joint distribution, we simply initialize our result with some initial values and start an iterative procedure that is guaranteed to converge to a true sample from the joint probability distribution. For each iteration, one of the random variables (RVs) (i.e. X_k) is chosen. All the other RVs are fixed to their current values, and a sample is drawn from the conditional probability distribution:

$$Pr(X_k = x_k|x_1, \ldots, x_{k-1}, x_{k+1}, \ldots, x_K)$$

We repeat this process until each of the K RVs has been chosen once. We then also repeat the whole process of choosing variables until the joint distribution $P(x_1, \ldots, x_K)$ does not change anymore. This state is known as *stationary distribution* or *thermal equilibrium*[37].
It can be shown that the gibbs sampling algorithm constructs a markov chain and that the stationary distribution of that markov chain corresponds to an approximation of the desired joint probability distribution [37, 18].

Algorithm 1 Gibbs Sampling

1: $U_k^{(0)} \leftarrow$ Initialize with values
2: **for** $k = 1, 2, \ldots, K$ **do**
3: generate $U_k^{(t)}$ from $Pr(U_k^{(t)}|U_1^{(t)}, \ldots, U_{k-1}^{(t)}, U_{k+1}^{(t)}, \ldots, U_K^{(t)})$
4: **end for**

In probabilistic graphical models and Bayesian networks, gibbs sampling is often used as an approximative technique to perform inference [7]. It can be applied efficiently when the topology of the network satisfies certain conditions.
This is particularly the case in *markov random fields* because there, the nodes (RVs) are only depending on the neighboring nodes. The conditional probability of an RV then becomes a function of the neighbors. Thus, we can use gibbs sampling in a parallel fashion because the units that are not neighbors are independent of each other and can be sampled at the same time. Especially, if the network is structured in *layers* and all nodes in one layer have no connections between them, the gibbs sampling algorithm can be applied very efficiently. In that special case, we can sample all conditional probabilities for one layer in parallel as the nodes from the other layers are observed and thus statistically independent from each other.

3.1.2 *Gradient Descent Algorithm*

Gradient descent is an iterative optimization algorithm to find the minimum in some differentiable function $E(x)$ with $x \in \mathbb{R}^N$. Usually, the surface of $E(x)$ is largely unknown, and explicit calculation of the extreme points of the function is not feasible.
So we wish to solve the following problem:

$$\min_{x \in \mathbb{R}^N} E(x) \tag{2}$$

In such a case, gradient descent starts with some initial random x and then follows the negative gradient of $E(x)$ by changing x by a small step in the direction of $\nabla E(x)$. This way, the algorithm slowly approaches a (local) minimum of $E(x)$ and converges when no further improvement is possible. Because the surface of E might be rapidly changing, the gradient has to be recalculated after following it for some time. It highly depends on the application how small this step size η can be. Small values for η follow the gradient more closely while large values for η speed up the convergence but can overshoot local minima. Starting at a random initial x^0 and for each iteration t of the gradient descent algorithm, we can solve the above minimization problem (equation 2) as:

$$x^{(t+1)} = x^{(t)} - \eta \nabla E(x^{(t)}) \tag{3}$$

In the context of most non-convex machine learning techniques, the function $E(x)$ is defined with respect to some training set. In that case, calculating the gradient means taking the average of gradients over all training points in the data set which is often not feasible.
Alternatively, one can use *online* or *stochastic gradient descent* which uses only one or a small batch of training points to estimate the gradient. This technique no longer follows the true negative gradient of E but it is shown to converge much faster [29]. Furthermore, the changes of x become more noisy with stochastic gradient descent which can even result in the overcoming of sub-optimal local minima [7].

Several extensions have been proposed to improve the often poor performance of the gradient descent algorithm, including *momentum*. With this method, the direction from the last step of the algorithm is incorporated into the current one which results in a behavior similar to a ball rolling downhill.
This is often useful when moving along "ridges" on the surface of E where normal stochastic gradient descent would "zig-zag" along the ridge. In such a case, momentum helps to find a stable direction and move down the "ridge" quickly.
Mini-batches are a trade-off between efficient computations (update

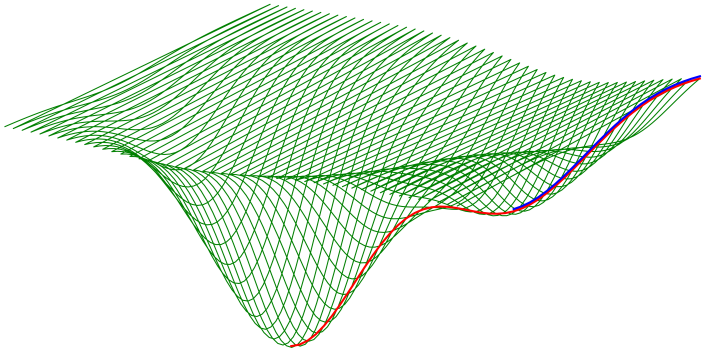

Figure 9: Gradient descent with (red) and without (blue) momentum. We can see that momentum allows the algorithm to jump over small saddle points in order to find a better minimum.

the model's parameters as often as possible) and mathematical correctness (update only after all the derivatives for all sequences are known and move in the direction of the "true" gradient). They process a small batch of training points at once but not the whole set.
There has been much research in the field and many scientific articles promote different opinions on the matter [57, 58] but in recent years, the mini-batch approach has been used extensively[2, 32, 50].
Many more improvements to gradient descent have been proposed but are not discussed here. See [46] for more information and variants of the gradient descent algorithm.

3.1.3 *Softmax Function & Categorical Distribution*

The softmax function calculates probabilities for different events when only raw counts of an event are available, and the categorial probability distribution can be used to represent the result from such a softmax function.
The probability for an event to occur should be high when the raw counts are high and low when the counts are low. Additionally, high counts should be over-represented such that the result of a softmax function is categorically distributed. This also implies that the probabilities for all events sum up to 1.
The softmax function achieves all that and can be defined as:

$$Pr(C_i|x) = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}} \tag{4}$$

where C_i denotes the classes and x is a vector containing the counts for each event.

The resulting distribution is called the *categorical probability distribution*, a generalization of the *binomial distribution*. Softmax functions

are often used in machine learning to classify an RV into discrete classes.
We use the categorical distribution extensively to sample from the result of the softmax function. This is often the case when we apply the probabilistic max pooling operation which is further described in section 3.4.1.

3.2 RESTRICTED BOLTZMANN MACHINES

We have presented the general idea of RBMs in chapter 2.3. The following chapter will focus on the mathematical derivations and details in training models. We present *contrastive divergence*, a training algorithm for RBMs that approximates the true partial derivatives to speed up learning and thus make training RBMs feasible in large scale applications.
An adaption that results in better approximations of the gradient is called *persistent contrastive divergence* and is presented here as well.
I will use the terms *gradient*, *derivative* and *partial derivative* interchangeable in this thesis. While all three of them describe the first order partial derivatives of some vector with respect to parameters θ, in literature the term *gradient* is most widely used.

3.2.1 *Formal Definition of RBMs*

Formally, an RBM can be interpreted as a maximum-likelihood problem. We can define a probability for both, the data and the hidden units. From there, it is possible to derive the parameters for exact maximum-likelihood learning. Let:

$$Pr(v,h) = \frac{1}{Z}\exp\left(-E(v,h)\right) \tag{5}$$

where E is the energy function defined as:

$$E(v,h) = -\sum_{i,j} v_i W_{ij} h_j - \sum_j b_j h_j - \sum_i c_i v_i \tag{6}$$

and Z denotes the partition function:

$$Z = \sum_{i,j} \exp^{-E(v_i,h_j)} \tag{7}$$

Z assures that the whole term stays in the range between zero and one and guarantees that $Pr(v,h)$ actually is a probability. However, the calculation of Z involves summing over all possible hidden states and therefore it is avoided to compute it explicitly. For instance, an

RBM with one hundred hidden units: To calculate Z, we would have to sum over 2^{100} different states, which is not feasible. There are methods to estimate the partition function but we will not go into detail here.
The energy function just counts the interactions between the visible and hidden units, weighted by the respective entry in W. Hence, we

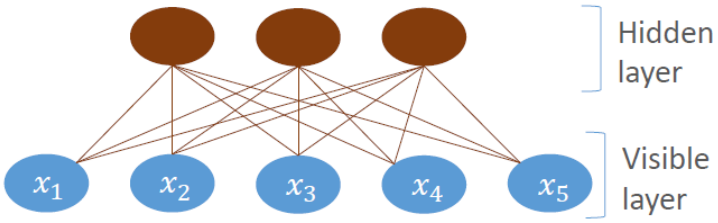

Figure 10: Schematic view of a RBM with five input units (blue) and three hidden units (red). Units in the same layer have no connections while every unit from one layer is connected to each of the units from the other layer.

can interpret the weight matrix as a function that chooses to give high importance to some interactions, while suppressing others. Formally, the independence of the random variables result in the following decomposition of the conditional probability:

$$Pr(v|h) = \prod_{j=1}^{N_h} Pr(h_j|v) \tag{8}$$

We can use this fact to calculate the probabilities of the hidden units to be activated by the data and vice-versa using equations 5 and 6:

$$\begin{aligned}
Pr(h_j = 1|v) &= \frac{Pr(h_j = 1, v)}{Pr(v)} \\
&= \frac{Pr(h_j = 1, v)}{Pr(h_j = 1, v) + Pr(h_j = 0, v)} \\
&= \frac{\frac{1}{Z}\exp\left(\sum_i v_i W_{ij} h_j + b_j h_j + \sum_i c_i v_i\right)}{\frac{1}{Z}\exp\left(\sum_i v_i W_{ij} h_j + b_j h_j + \sum_i c_i v_i\right) + \exp(\sum_i c_i v_i)} \\
&= \frac{\exp(\sum_i c_i v_i)\exp\left(\sum_i v_i W_{ij} h_j + b_j h_j\right)}{\exp(\sum_i c_i v_i)\left(\exp\left(\sum_i v_i W_{ij} h_j + b_j h_j\right) + e^0\right)} \\
&= \frac{\exp\left(\sum_i v_i W_{ij} h_j + b_j h_j\right)}{\exp\left(\sum_i v_i W_{ij} h_j + b_j h_j\right) + 1} \\
&= \sigma(\sum_i v_i W_{ij} + b_j)
\end{aligned}$$

The conditional probability for the visible layer, given the hidden one can be computed similarly. We get the following two equations for the conditional probability when we use equation 8:

$$Pr(h|v) = \prod_{j=1}^{N_h} \sigma(\sum_i v_i W_{ij} + b_j) \tag{9}$$

$$Pr(v|h) = \prod_{i=1}^{N_v} \sigma(\sum_j W_{ij} h_j + c_i) \tag{10}$$

These two equations can now be used to perform *Block Gibbs Sampling* to obtain samples from the joint probability distribution over a data point. This can be done by applying equations 9 and 10 in an alternating fashion and sampling from the distributions in each step.
That is, we first compute the activations in the hidden layer and then reconstruct the visible layer from there.

Now we have all the components together to perform *maximum-likelihood learning* on the data, where we try to maximize the parameters to fit the data best.
In a maximization procedure, it is equivalent to optimize a function or the logarithm of it because the logarithm is a monotonically increasing function which has the same minima/maxima as the original objective in the problem.
Let θ be the set of parameters with $\theta = \{W, b, c\}$. We then denote $\mathcal{LL}(\theta|\mathcal{D})$ to be the data log-likelihood, that is the natural logarithm of the data likelihood $\mathcal{L}(\theta|\mathcal{D})$.
Assuming that the samples were all drawn from the same underlying probability distribution and being independently and identically distributed (i.i.d), we can make the following derivations:

$$\mathcal{L}(\theta|\mathcal{D}) = \max_\theta \big(P(v,h)\big) \tag{11}$$

$$\mathcal{L}(\theta|\mathcal{D}) = \max_\theta \prod_{i=1}^{N} \big(P(v_i,h)\big) \tag{12}$$

$$\mathcal{LL}(\theta|\mathcal{D}) = \max_\theta \sum_{i=1}^{N} \log\big(\frac{e^{-E(v,h)}}{\sum_{i,j} e^{-E(v_i,h_j)}}\big) \tag{13}$$

$$\mathcal{LL}(\theta|\mathcal{D}) = \max_\theta \sum_{i=1}^{N} \big(\log \sum_h e^{-E(v_i,h)} - \log \sum_{v,h} e^{-E(v,h)}\big) \tag{14}$$

We can see that the log-likelihood is the summation of two similar terms.
However, they are very different due to the summation over v in the second term. For the first term we have to sum over the energies for the data and hidden units. We remember from the previous section that this means actually just counting the interactions between V and H, weighted with W. The second term, however, is more complex

as it involves the joint probability distribution over v and h. This is where we need the Block Gibbs Sampling mentioned earlier in order to get a sample from that distribution.

3.2.2 *Learning in RBMS*

To learn the parameters of an RBM, we have to calculate the derivative of the (log-) likelihood function. Because the exact gradient cannot be calculated analytically, we will use the *gradient descent* algorithm from chapter 3.1.2 to iteratively increase the likelihood of the data.

Following the derivations of Fischer et al in [16], we can write the derivative of the log-likelihood function in the following way:

$$\frac{\partial \mathcal{LL}(\theta|v)}{\partial \theta} = \frac{\partial}{\partial \theta}\Big(\ln \sum_h e^{-E(v,h)}\Big) - \frac{\partial}{\partial \theta}\Big(\ln \sum_{v,h} e^{-E(v,h)}\Big) \tag{15}$$

$$= -\sum_h \frac{e^{-E(v,h)}}{\sum_h e^{-E(v,h)}} \frac{\partial E(v,h)}{\partial \theta} + \sum_{v,h} \frac{e^{-E(v,h)}}{\sum_{v,h} e^{-E(v,h)}} \frac{\partial E(v,h)}{\partial \theta} \tag{16}$$

$$= -\sum_h Pr(h|v) \frac{\partial E(v,h)}{\partial \theta} + \sum_{v,h} Pr(v,h) \frac{\partial E(v,h)}{\partial \theta} \tag{17}$$

The last line of our calculation used the following equality:

$$Pr(h|v) = \frac{Pr(v,h)}{Pr(v)} = \frac{\frac{1}{Z} e^{-E(v,h)}}{\frac{1}{Z} \sum_h e^{-E(v,h)}} = \frac{e^{-E(v,h)}}{\sum_h e^{-E(v,h)}} \tag{18}$$

Remember that our hidden variables h_i are in fact random variables with binary outcomes. Hence, the derivative in equation 17 is the difference between two expected values. In the first term of 17, we sum over all RVs in H and multiply the probability of the unit being activated with the derivative of the joint energy function w.r.t. θ.
Earlier in section 3.2.1, we have seen that the energy mainly counts the pairwise activations of visible and hidden units. So the derivative of the energy with respect to W gives us:

$$\frac{\partial E(v,h)}{\partial W} = \frac{\partial}{\partial W} v^T W h = v h^T \tag{19}$$

Thus, the first term of our partial derivative is simply the expected value of the pairwise interactions between visible and hidden layer.

However, the second term of the partial derivative is much more complex, as it involves the joint probability instead of the conditional probability. It can be written as the same expected value as the first

term, but under the models probability distribution.
Thus, we can write the gradient as:

$$\frac{\partial \mathcal{LL}(\mathcal{D}|\theta)}{\partial \theta} = -\mathbb{E}\left[vh\right]_{data} + \mathbb{E}\left[vh\right]_{model} \tag{20}$$

This rather convenient notation could already be seen in equation 14 because of the summation over only the hidden units in the first term, and the summation over hidden and visible units in the latter term.

Before we try to calculate the derivative of the models expected value, a more detailed look at equation 20 will give us an intuition of what the RBM is trying to learn. In the first phase, dubbed *positive phase*,

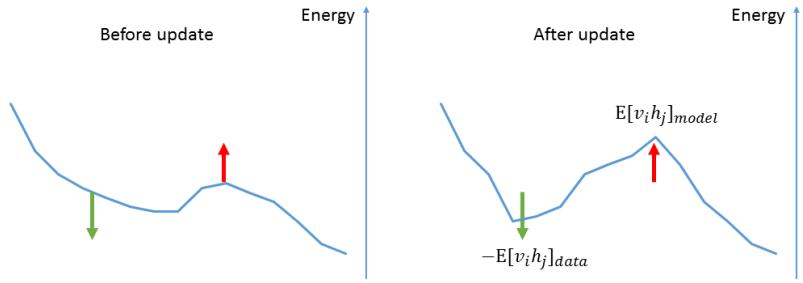

Figure 11: An example 1-D energy surface before and after the update of the parameters. The energy decreases where the datapoint is located but increases at positions where the model has high energy.

the energy is pushed downwards for the data, while in the *negative phase*, the energy is pulled up for the joint probability.
This means that the space where the data is located becomes more probable, while the areas of the energy function where other possible data (but data that is not in the training set) is located, increases. This way, combinations of visible units that do not (or rarely) occur in the training data will become less probable.
With an RBM that is fully trained, these two phases are equal and thus cancel each other out.

The first part of equation 20 can be computed easily, using equation 9. It is simple counting how often a specific visible unit was activated together with a hidden unit.
To calculate the expected value for the model, as we have seen earlier, we have to sample from a very complex probability distribution.
Luckily for us, obtaining a sample from a complex distribution is often less complicated than calculating the distribution itself. In our case, we can use *block gibbs sampling* to obtain a sample from the models distribution.
This approach works in cases where it is difficult to obtain samples from a joint distribution, but it is relatively easy to get samples of a

conditional distribution. As it turns out, our setting is a somewhat perfect application for gibbs sampling because we can easily get the conditional probabilities for the hidden layer, given the visible layer and vice versa.
Thus, we can first calculate the probabilities for H, given V, and then do the opposite to obtain a sample of V, given H.

Sampling theory promises us, that when we repeat this alternating sampling for a long time, we will eventually reach the state of thermal equilibrium, or the *stationary distribution* of the gibbs chain.
This stationary distribution is then a sample of the true joint probability distribution $Pr(v, h)$. The only problem that remains is that it can take a very long time for the chain to settle in thermal equilibrium and this renders the classical log-likelihood based RBM training algorithm practically useless.

3.2.3 *Contrastive Divergence*

Obtaining a true sample from the joint probability distribution $Pr(v, h)$ is hard but we only need it to calculate the gradient from there. To estimate the direction in which the gibbs chain is going from a data point, it seems to be sufficient to calculate one step in the gibbs chain to get an approximation that gives reasonable results in practice.[21]
This shortcut to just calculate one or two steps in the gibbs chain is called *Contrastive Divergence* and provides the first efficient algorithm to train RBMs [23].
When we only run the gibbs chain for one iteration, the process is

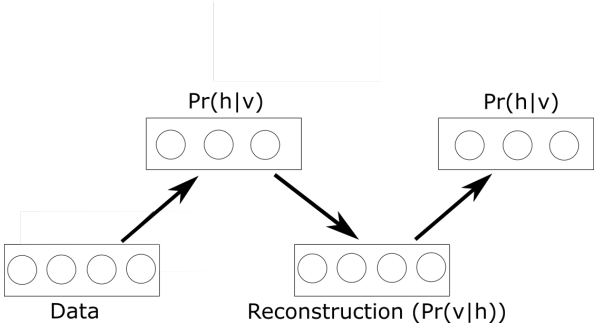

Figure 12: An example of an RBM with two hidden and 3 visible units. The gibbs chain is only run for one iteration.

called *CD 1* (as depicted in figure 12). It is often useful to start with only one iteration and increase the number of steps gradually.[21]
To interpret the meaning of the gibbs chain's length, we can imagine the models *energy surface*. The data-point is located somewhere on it and from there we start running the chain. With each step, we get

away from the data-point and follow the gradient on our way. When we only use *CD 1*, we cannot reach far from our data-point with the gibbs chain. The energy gets raised somewhere in close proximity to the data-point.
But reaching further might be necessary because we can have an energy minimum somewhere far from the data point and the model tends to assign a high probability to that point. The training process would normally gradually increase the energy at this point far away from the training data and thus decrease its probability. But when we can't reach it with contrastive divergence, this increase of the energy never happens.
When we now see new data for which we want to calculate the posterior probability and this new data is located somewhere at this energy minimum that we couldn't reach with our gibbs chain during training, we get high probabilities for our new data.
However, this should not be the case as this new data is nothing alike the training data (otherwise it would have been closer to the other data points).

3.2.4 *Persistent Contrastive Divergence*

In the previous paragraph, we saw that contrastive divergence is only increasing the energy at points on the error surface which are relatively close to actual data points. When the test data is very different from the training data, we might assign low energy (and thus high probability) to this data, even though it does not belong to the probability distribution where the training data originated from.
This is undesired behavior and can be overcome using an algorithm very similar to contrastive divergence, called *persistent contrastive divergence*.
The main idea behind this technique is to not start the gibbs chain from the data point, but rather from a point on the energy surface which is known to be at an energy minimum from the past. As it happens, such points are exactly those, for which we rose the energy when we were looking at the previous data point [53].
That means, we simply store the result from the gibbs chain for our current data point and then reuse this as beginning for the next gibbs chain.

The points on the energy surface are called *fantasy particles* and are located in areas to which the model would assign high probabilities. They can be interpreted as the models convictions. Since we initialize an RBM with random initial weights, these convictions might be wrong, and therefore we try to increase the energy in these areas.

A more philosophical interpretation of the negative phase of the energy gradient is given by Geoffrey Hinton in [22]. He sees the whole negative phase (also in standard contrastive divergence) as an "unlearning" taking place. This way, similar to humans dreaming, the model would focus on correlations in the data that are important and often occur, while "forgetting" other connections which seem less important.
However, as described previously, the negative phase also makes sense in a mathematical interpretation of the process.

3.3 CONVOLUTIONAL RBMS

We now have a model for which we can maximize the data likelihood. But until now, this model has a few drawbacks that we need to tackle. The first and most important one is the problem of having *too many parameters to tune*. Imagine the application of RBMs to DNA sequences. We would have at least one visible unit for each letter of the sequence, which is only possible when modifying the original (binary) algorithm to deal with more than only ones and zeros.
Because of the fully connected nature of an RBM, we have $N_v \times N_h$ many connections, where N_v is the sequence length and N_h is the number of categories we want to find. Learning this matrix means learning many parameters which requires a huge amount of data to avoid overfitting.
The second problem we have to tackle is the *lack of translation invariance* or at least equivariance. Finding certain combinations of letters in a DNA sequence repeatedly is not achieved by the standard RBM at all.
And lastly, the *interpretability* of the model is important for our application. But there is no meaningful interpretation for the hidden layer in the normal RBM setting.
To overcome these problems, the image recognition community has been using convolutional neural networks very successfully in the past and also for RBMs, a convolutional counterpart has been developed by Ng et al. in [32].

The convolutional variant of the restricted Boltzmann Machine is very similar to the extension of neural networks to convolutional neural networks (cNNs). To have fewer parameters when the input dimensions get large, the layers are no longer fully connected.
Instead, the weights form matrices of predefined size and "slide" over the input data. Thus, the same weights are used at different positions in the input. When dealing with DNA sequences, the weight matrices (sometimes called *kernels*) correspond to *motifs*. The hidden layer

then can be seen as motif hits for a particular kernel.

3.3.1 *The 2-D Convolution for images*

Convolution for images is a filtering operation of two matrices that are multiplied with each other in a special way. In a convolution operation, denoted by $*$, we have two different kinds of matrices. The first is the image, which in the cRBM setting will be the data, denoted as I. The second kind of matrix is the so-called *kernel* which is typically much smaller than the image and denoted by K.
The kernel is first turned horizontally and vertically. Then for each pixel in the image, the kernel is multiplied element-wise with the kernels center being located at the chosen image coordinate.
Types of convolutions differ in how they deal with the images edges. In a *valid* convolution, the multiplication is only performed for those image pixels that are far enough "inside" the image such that the kernel can be applied whereas in *padded* convolutions, the image is padded with zeros or ones. Let $I \in (M \times N)$ and $K \in (U \times V)$. Then

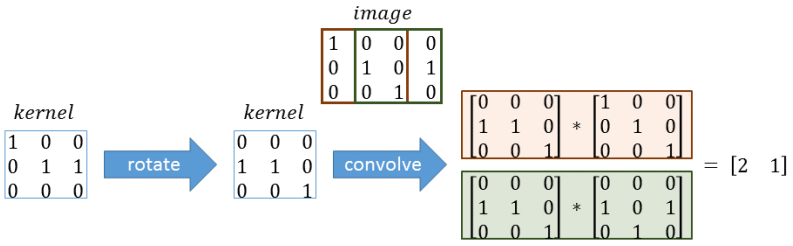

Figure 13: A schematic view of the convolution operation. The kernel is first flipped and then applied to the image in a "sliding" fashion.

$C = V * K$ is defined as:

$$C_{x,y} = \sum_{u=0}^{U} \sum_{v=0}^{V} I_{x-u+1,y-v+1} K_{u,v} \tag{21}$$

The resulting matrix $C \in (M - U + 1, N - V + 1)$ is smaller than the original image. We also directly see that, when both I and K have the same number of rows, C becomes a vector. We use this property in our setting with DNA sequences later.

Another operation that will become necessary for the definition of cRBMs is the *cross-correlation* filtering operation. It is similar to the convolution but flips the kernel before applying it. It is thus the same operation as "sliding" the kernel over the image **without** flipping the kernel at all.

3.3.2 *Definition of the cRBM*

With the convolution operation at hand, we can now formulate the extension of the conventional RBM to the convolutional RBM. To achieve that, we only have to modify the weight matrix W. Instead of simply connecting V and H with a fully connected matrix, we now have K kernels that are convoluted with V.
Each of these filtering operations produce a vector h_k and thus $H \in \{0,1\}^{K\times(N-M+1)}$ is a matrix with rows h_i.
With these modifications, we can write the conditional probabilities in the following way by derivations similar to those in [16]:

$$Pr(h_j^k = 1|v) = \sigma(v_i * \widetilde{W}^k + b_j) \tag{22}$$

$$Pr(v_i = 1|h) = \sigma(\sum_k W^k * h^k + c) \tag{23}$$

where $\widetilde{W}_k$ indicates that the kernel is flipped horizontally and vertically. With these conditional probabilities at hand, we can perform up- and down-sampling. To apply the contrastive divergence algorithm of chapter 3.2.3, we only have to calculate the gradient of the data-log-likelihood.

3.3.3 *Learning in cRBMs*

In cRBMs, the energy function is defined slightly different than in the standard RBM scenario. The interaction between the visible and hidden layer can be written as filtering operation and thus, the energy function becomes:

$$E_c(v,h) = -\sum_{k=1}^{K} h^k(\widetilde{W}^k * v) - \sum_{k=1}^{K} b_k \sum_i h_i^k - c^T \sum_i v_i \tag{24}$$

Calculating the derivative of this new energy function yields us:

$$\frac{\partial E_c(v|\theta)}{\partial W_k} = \sum_k Pr(h_k|v)\frac{\partial E_c(V,h_k)}{\partial W_k} \tag{25}$$

And for the derivative of the joint energy, we get:

$$\frac{\partial E_c(V,h_k)}{\partial W_k} = -(v * h_k) \tag{26}$$

This means, we can express the derivative of the joint energy function as a convolution operation. Thereby, we can re-formulate the calculation of the derivative from section 3.2.1 and use the derivations made for the standard RBM with minor adaptations.
By using the energy function from equation 24 and the derivative

of the log-likelihood from equation 17, we can re-write the expected value for the two phases similarly to equation 20.

$$\frac{\partial \mathcal{LL}(\mathcal{D}|\theta)}{\partial \theta} = -\mathbb{E}\left[v * h\right]_{data} + \mathbb{E}\left[v * h\right]_{model} \tag{27}$$

This result is again surprisingly simply and very convenient. We can express almost all of the operations needed for learning as convolutions for which many libraries provide highly optimized code. They can also make use of massively parallel hardware architectures and especially of GPGPUs.

3.3.4 *Applying cRBMs to DNA sequences*

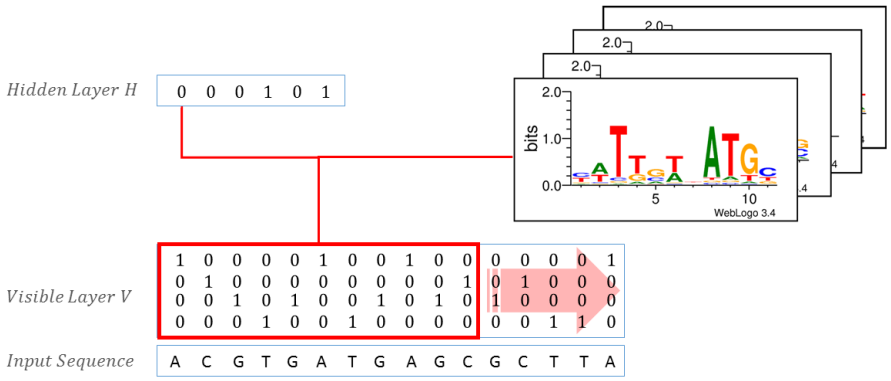

Figure 14: Convolutional layer of a cRBM. The convolution produces a smaller hidden layer H that contains the motif hits. We see that the DNA sequence is encoded as one-hot-matrix. Furthermore, the kernel is equivalent to a position-weight-matrix and can be visualized as web-logo.

We mentioned in chapter 3.3 that the transition from RBM to cRBM would also provide us with a solution to the problem of DNA sequences to binary input. This solution is called the *one-hot-representation*. Each of the nucleotides from the DNA alphabet gets a four-dimensional vector assigned, and the one-hot-representation is simply a matrix $V \in (4 \times N)$, where N is the length of the sequence. The mapping from letter to vector is defined as follows:

$$A \mapsto \begin{bmatrix}1\\0\\0\\0\end{bmatrix}, C \mapsto \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, G \mapsto \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, T \mapsto \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

The resulting matrix has in every case four rows only. When we define all kernels also to be of four rows and M columns, then the

hidden layer becomes a vector per kernel and is now much easier to interpret than before.
The kernels now correspond to *motifs*, while the hidden layer contains motif hits. A motif is a short and usually repeating pattern in DNA sequences. It is assumed that many motifs have biological functions, such as being a binding site for a certain DNA binding protein.
This suddenly makes the cRBM an interpretable model in which the weights correspond directly to position-weight-matrices and the hidden layer corresponds to the presence of a particular motif in the sequence.
When we now apply probabilistic max pooling to the hidden layer, we get probabilities for each of the units in H to be active or not while neighboring units are combined to decrease the size of H.

To summarize, by going from a standard RBM to convolutional RBMs we obtained a model with fewer parameters. Also, we gained translation equivariance. This means that the hidden layer has a slightly different structure than with conventional RBMs. The hidden layer now corresponds to a matrix in which the rows represent the different kernels, and the columns represent nucleotides on the DNA to which the motif detector was applied.
However, we have the problem of over-completeness now. Because we usually want to use more than one motif in the cRBM, the hidden layer is much larger than the input. This is a general problem because the model might just learn trivial representations of the data and not features that generalize well. To develop a model that does not overfit the data, we still have to apply some tweaks to enforce sparsity in H.

3.4 ENFORCING SPARSITY

As we have seen, it is crucial for successful learning to enforce sparsity in the hidden layer. This will not only result in better reconstructions of the data but especially in better representations of the sequences in H.
In the following section, I will give an overview of different methods for sparsity enforcement that we implemented.

3.4.1 *Probabilistic Max Pooling*

Pooling is a very popular technique for sub-sampling from a convoluted image in image recognition. To move from very specific and small regions in the image, it is typically desired to have a pyramidal structure and larger contexts in higher layers of any convolutional

neural network (cNN). Pooling sub-samples an image by a predefined size and thus makes it smaller. The result from the convolutional layer is therefore divided into distinct regions. From those regions, only one value "survives" the pooling step. This may be the maximum (dubbed *max pooling*) or the mean (dubbed *average pooling*).
The resulting image may only be one quarter of the size of the origi-

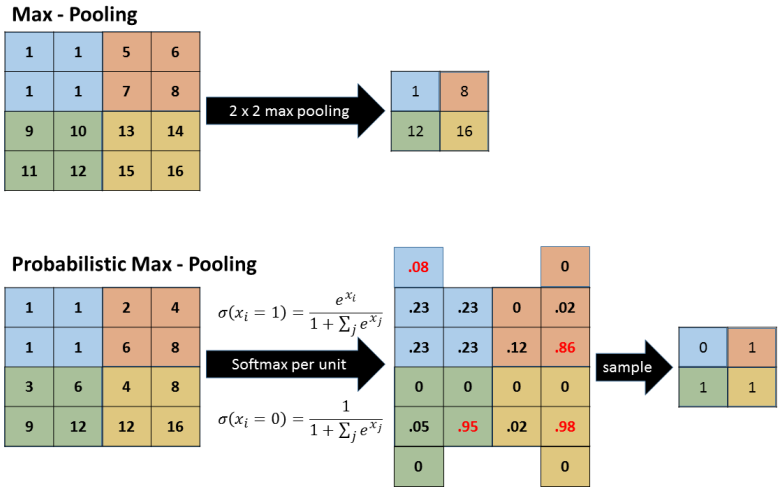

Figure 15: Traditional max pooling and probabilistic max pooling in comparison. We see that the traditional approach is deterministic while the probabilistic approach only samples from the softmax distribution within each unit. The additional units represent that there is no activation at all in the pooling unit. Note that the numbers in red are chosen by the sampling step.

nal (when using pooling with bins of size 2×2).
However, that approach would fail drastically for cRBMs due to the stochastic environment. When we only calculate probabilities, taking the maximum probability will result in deterministic behavior which is unwanted for our model.
Lee et al. [32] described an algorithm, called *probabilistic max pooling* that calculates the probability for a unit in H to be activated, given the other units from the pool. Then sampling takes place and randomly draws one of the units or none of them.
This way, we have at maximum one activated unit per cell while maintaining the probabilistic nature of the RBM. Technically, a softmax distribution is calculated for each of the cells and the probability for none of the units to be activated can be written as:

$$Pr(P_i^{(k)} = 0|v) = \frac{1}{1 + \sum_\alpha e^{Pr(h_\alpha^{(k)}|v)}}$$

while the probability for the other units is:

$$Pr(P_i^{(k)} = 1|v) = \frac{e^{Pr(h_i^{(k)}|v)}}{1 + \sum_{\alpha} e^{Pr(h_\alpha^{(k)}|v)}}$$

Note that P does not indicate a probability but the pooled layer that we can imagine as being on top of the hidden layer H. α stands for the index within a cell of pooled units.

3.4.2 *Regularization*

We can enforce sparsity of the hidden layer not only by pooling together units. Instead, we can constraint adjacent regions to only have a certain number of hits, and we can also pool vertically to enforce competition between different kernels due to the locality properties of pooling.
While this is very useful to get clearer motifs, we still lack a possibility for regularizing our model to only have a particular global ratio of motif hits.

Therefore, we introduced a sparsity constraint that acts as regularizer. It forces the partial derivatives of the probabilities to meet a certain target. This way, our objective is to maximize the likelihood of the data while forcing the weights to only permit ρ percent of the hidden units to be activated. More formally, we get:

$$Reg(V) := \sum_{k=1}^{K} \left(ReLu(\mathbb{E}[h^{(k)}|V] - \rho) \right)$$

The regularizer is added to the gradient calculation as Lagrange multiplier just like in any other machine learning setting. We then get the new update rule for the gradient descent algorithm:

$$W^{(k)} = W^{(k)} - \eta \left(\frac{\partial \mathcal{LL}(\mathcal{D}|W^{(k)})}{\partial W^{(k)}} - \lambda \nabla_{W^{(k)}} Reg(V) \right) \tag{28}$$

3.4.3 *Initializing the weights*

A problem that may arise when using regularization of the objective function (the *data log-likelihood* in our case) is that the gradient might only follow the regularization term at the beginning of the training. This can slow down the learning significantly and might also be detrimental to the overall result as the quality of the learning strongly depends on the initial configuration of an RBM.

Thus, the initial weights have to be chosen very carefully. In our approach, we tried to initialize the weights randomly from a gaussian distribution but took great care that the activation in H is initially already sparse. This means we want the sum of activations to be approximately the same as ρ.
Drawing a matrix of standard gaussian numbers which is then used in a 1-D convolutional setting is equivalent to drawing as many RVs from it as there are columns in the matrix. Since the addition of gaussian *RVs* is the same as adding their mean and variance, we draw from the following distribution:

$$\rho \approx \sum_{h \in H} Pr(h_k|v)$$
$$\rho \approx \sum_{h \in H} softmax\Big(corr(v, \mathcal{N}(0,k))\Big) + b_k$$

where k is the number of columns (or *k-mers* when in our motif detection setting). Because of the standard normal distribution, the mean doesn't shift and the variance simply adds by one with each additional random variable. Our goal is now to set the models bias b_k in

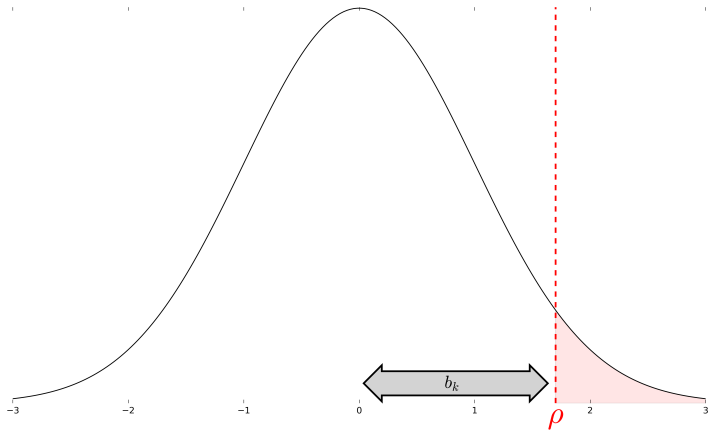

Figure 16: The desired threshold ρ. We want to compute b_k in order to know how the initial intercept of the model should be set. The samples then only come the shaded area.

a way that only ρ percent of the units in H are active. The *quantile* of a gaussian is the inverse function of the cumulative distribution function. It can be interpreted as how far we have to move a distribution in order to get a certain amount of the population to be above a certain threshold.
Thus, we get the following equation on how to set the initial bias of our model:

$$b_k = \Phi^{-1}(\rho, \mathcal{N}(0,k))$$

This method does not necessarily yield very accurate activation in H because the data is not uniformly distributed. But it gives a good enough estimate such that the learning starts already during the first couple of iterations.

4

EXPERIMENTS & RESULTS

In the following chapter, I will present the results from several experiments. For this thesis, we focused on the three main problems defined in chapter 1 when conducting the experiments.
The implemented cRBM model contains all of the extensions to the original algorithm that were presented in chapter 3, including persistent contrastive divergence, probabilistic max pooling, and regularization.
This chapter is organized as follows: First, we will present the training algorithm and details of the implementation. We will then look at the results from three experiments. We monitored different aspects during the training to be able to answer the following questions:

1. When learning features of DHSs in an unsupervised way, does the cRBM recover known transcription factor binding sites?

2. Can the cRBM be used in a supervised setting to classify tissue specific DHSs by looking at the trained probability distributions?
   How well can we classify with this generative model?

3. Is it possible to train a cRBM model on a control data set from PAR-CLIP experiments and then find biases?
   Could we then further use this trained model to classify sequences into background and signal, thus identifying sequences that were pulled by PAR-CLIP erroneously?

4.1 IMPLEMENTATION & HARDWARE DETAILS

I decided to use the Theano framework [6] and the Python programming language to be able to train the model in an efficient way while having the means of a high-level programming language at hand.

Python is widely used for scientific computing and has thus many packages for data reading, processing and visualization already implemented. Also, some packages for biological applications, like the generation of web-logos [12] and reading fasta-files [9] were freely available, making Python the programming language of choice.
On the other hand, Theano allows for the easy and highly optimized execution of code on a GPGPU which can vastly accelerate execution of the training. Especially, convolutions can be calculated massively in parallel by modern GPGPU architectures and yield speedups of 25-50 times when using latest drivers for the graphics card together with the CUDA language. Since training involves especially many convolutions, using the NVIDIA CUdnn library can speed up the process again by up to 70 times.[8]

In Theano, programming is done by defining a *computational graph* at first by using special operations that belong to the framework. This graph is later compiled to a function that can then be called just like a normal Python function. The advantage of that approach is that Theano can optimize the graph for different objectives, such as numerical stability or efficiency. Furthermore, the framework can make use of GPUs when one is available, but it still works when there is none.
Numerical stability was necessary for our project because of the vast use of softmax-functions prior to sampling. For the experiments, we had a server with two Nvidia GTX 980 graphics processors available. However, at the time when we conducted the tests, Theano did not offer capabilities of using two GPGPUs at the same time by sharing memory between them. Thus, we only took advantage of one of the cards but were able to train multiple models at the same time. The second major benefit of Theano lies in the automated calculation of derivatives which is a huge advantage when using standard gradient based methods. However, due to the gibbs sampling part in restricted Boltzmann machines, this option was not available for us. For such a complex partial derivative to be computed automatically, Theano would have to know when the gibbs sampling reached a stationary distribution which is hard and very specialized.
Furthermore, the shortcut of contrastive divergence no longer calculates the exact partial derivatives but rather an approximation to them. As a consequence, I could not benefit from Theanos capability to calculate gradients automatically but had to implement contrastive divergence myself.

The code consists of several applications, each for a different purpose. While the main functionality for training and evaluating cRBM models on sequences is located in one file, so called observers can be added to monitor the training progress. These are small classes that

implement an interface and score the model at a given time point during training. Observers can, for instance, calculate the reconstruction error of the model, given a set of parameters. Monitoring this gives us valuable information on overfitting that might occur during training.
This way, a modular structure of the code is maintained, and new observers can easily be implemented when desired, and the implementation of the cRBM itself remains separate.

4.2 THE GENERAL TRAINING PROCEDURE

Learning the kernels or motifs of a cRBM requires sampling the hidden layer first with a data vector clamped to the network as described by equation 22. The work-flow to calculate the hidden layer is depicted in figure 17.

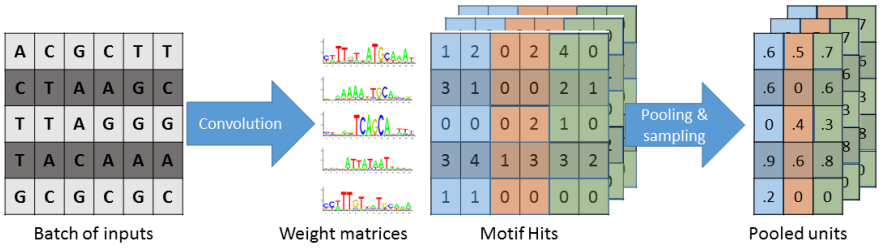

Figure 17: The general work-flow of a forward pass in our cRBM model. The input batches are first convoluted with the weight matrices, forming multiple layers of activation. The result is then transformed into probabilities with *probabilistic max pooling*. From there, H can be sampled.

The backward pass can be thought of as reconstruction of the original data (the sequences) from its representation (linear combination of motifs). Equation 23 describes how the visible layer can be reconstructed from the hidden layer. Analogously to the forward pass, this computation can be described as a convolution operation and thus can be calculated efficiently on a GPU. Figure 18 depicts the work-flow of the backward pass.

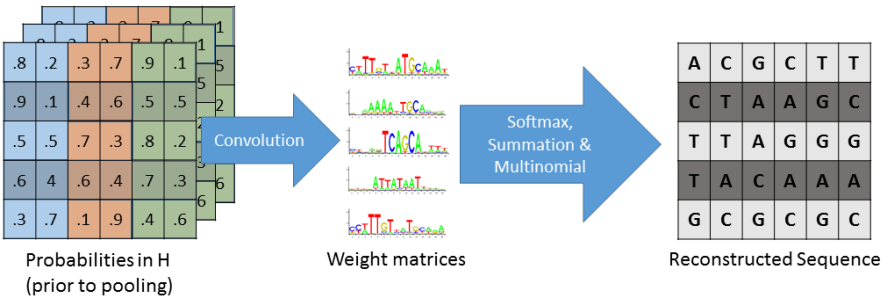

Figure 18: Workflow of the backward pass. The probabilities in H are convoluted with the weight matrices to form a multinomial distribution over the letters from which a sequence is then sampled.

As already stated in chapter 3.3.2, the derivative of the conditional energy function can be expressed as a convolution operation. The partial derivatives of the data expected value can be computed as soon as H^0 is available, that is when the upward pass was calculated. Algorithm 2 shows the whole training procedure in detail. Once the statistics for the data - the *positive phase* - are computed, we run the gibbs chain for k steps to obtain H^k iteratively applying the forward and backward pass. The *prob_max_pooling* operation partitions a matrix into small units of a fixed size and calculates a softmax in them as described in section 3.4.1. The *categorial* operation samples from a matrix, interpreting each column as different states of one random variable. Given a matrix $M \in \mathbb{R}^{m \times n \times k}$, we then get $m \times n$ samples, reducing the last dimension of the matrix. Finally, the $softmax$ operation computes probabilities from raw counts per column (over the different letters of the alphabet) according to equation 4.

Algorithm 2 CRBM Training

1: $H^{(0)} \leftarrow prob\_max\_pooling(V * \widetilde{W} + b)$
2: $H^{(0)}_{sample} \leftarrow categorial(H^{(0)})$
3: $Grad_{data} \leftarrow V * H^{(0)}$ ▷ Calculate data gradient
4: **for** $i = 1, 2, \ldots, CD_k$ **do** ▷ Contrastive Divergence Iterations
5: $V^{(i)} \leftarrow softmax(\sum_{k=1}^{K} H^{(i-1)} * W + c)$
6: $V^{(i)}_{sample} \leftarrow categorial(V^{(i)})$
7: $H^{(i)} \leftarrow prob\_max\_pooling(V^{(i)}_{sample} * \widetilde{W} + b)$
8: $H^{(i)}_{sample} \leftarrow categorial(H^{(i)})$
9: **end for**
10: $Grad_{model} \leftarrow V^{(i)}_{sample} * H^{(i)}$ ▷ Calculate model gradient
11: $W \leftarrow W + \eta(Grad_{data} - Grad_{model})$

We designed our training algorithm to always process multiple sequences at once in so-called *mini-batches*. The solution is updated after one mini-batch has been processed. The size of each mini-batch depends on the number of data points available and is recommended to rank between 20-100 data points at once [21]. The sequences should be randomly ordered before training because mini-batches of similar sequences could guide the derivatives in a direction which is not necessarily corresponding to the direction of the actual gradient. The data is probably ordered due to its pre-processing and DHSs in similar loci are grouped together. To avoid this, a random shuffling should always be applied to any training data when one of the assumptions of the algorithms is that the samples are independent and identically distributed (i.i.d.).

Training consists of multiple *epochs* and during one epoch all sequences are shown to the model once. In our experiments, we typically trained the model for approximately 300 epochs and observed a saturation in most of the cases.
The reconstruction rate and free energy of the model were measured after each epoch for a given training and test set while the test set was never shown to the model. The reconstruction rate denotes the average number of nucleotides that are correctly retrieved after one iteration of the gibbs chain with the current parameters. Figure 19 shows the reconstruction rate and free energy for an exemplary training.
The free energy is a measure that is similar to the negative log-likelihood (NLL) of the data, given the parameters of the model. This value should drop if the learning is successful. The free energy is the most direct measure of the real data-likelihood that we can get because the partition function (Z) is not easy to estimate. Recall the definition of Z as given by equation 7. To know the correct value of Z, we have to sum over all possible combinations of v_i and h_i for all i, j in the sequence and the hidden layer. This is not feasible for larger sequences. While methods exist to estimate the partition function, for relative comparisons we are only interested in the NLL. In such an equation, the partition function does not matter as shown by equation 29.
Since both values were monitored for the test and a sub-sampled training set, detecting overfitting would have been possible. However, we never observed such a divergence between test and training values.

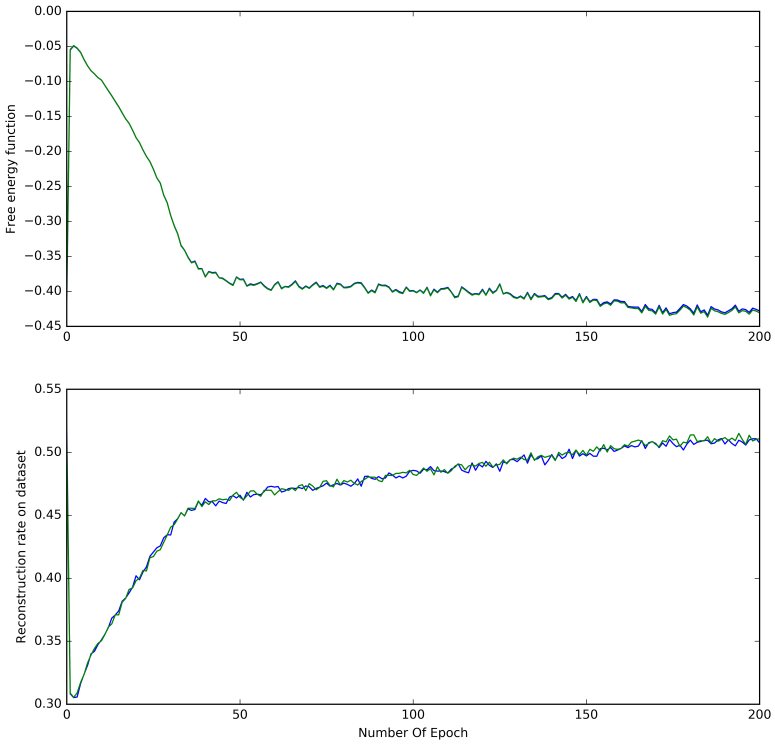

Figure 19: A typical result from monitoring reconstruction error and free energy during training. The initial rise (or drop respectively) of the free energy could originate in the regularization where the algorithm only tries to fulfill the sparsity constraints and does not focus on minimizing the free energy yet.

4.3 MOTIF DETECTION WITH THE CRBM

In the first experiment, we wanted to assess whether the cRBM was able to successfully reconstruct motifs of transcription factor binding sites from data where such binding sites were known.
We trained the convolutional restricted Boltzmann Machine on stem-cell DHSs that were extracted from CHiP-seq experiments in 2011. The data is online as part of the ENCODE project [11]. Known transcription factors for eukaryotic stem cells include *oct4* and *sox2* and their binding preference is recorded in the JASPAR database [38].
During the training, we monitored the reconstruction rate and free energy after each epoch as depicted in figure 19. Indications for successful learning are a decrease in the free energy and a gradual increase in the reconstruction rate. It is evident, however, that the reconstruction

rate will never approach 100% due to the sampling noise and the fact that the cRBM is supposed to only learn over-represented patterns in the sequences.

We also monitored several other statistics during learning, such as the average information content in the motifs or the average number of hidden units activated by a sequence. While we could observe a rising information content (IC) during training, this value does not tell us much because it can temporarily decrease for the model to find better motifs.
After the training, we visualized the motifs as web-logos and compared them to known motifs from the JASPAR database.

4.3.1 *Detecting motifs in eukaryotic stem cells*

When training the cRBM on eukaryotic stem cell data, we were able to detect the *oct4* transcription factor binding site which is known to bind in stem cell DHSs. Repeating the training several times with different random initializations always yielded similar results, indicating both, a strong presence of the motif and a certain robustness of the algorithm. This is important because RBMs, like all kinds of neural networks, solve a non-convex optimization problem and converge to local minima. Therefore, they strongly depend on the initialization.
The choice of the right intercept (or bias b) was crucial, however, to get reasonable results when using regularization. A technique to approximate an intercept for a given desired sparsity is explained in detail in chapter 3.4.3.

The convolutional RBM algorithm only detects fixed length motifs (the kernel length has to be the same for all kernels for computational efficiency).
Furthermore, we are not guaranteed that the pattern is not split up between multiple motifs which would make it hard to see even though it has been learned successfully. Parts of a motif, especially those that have a low IC can even be considered noise because they are not over-represented in the data due to the variability of the protein to bind slightly different nucleotides at a given position.
A stacked approach of multiple cRBMs upon each other might fix this problem since, in such a case, we would look out for interactions and patterns between the motifs we found in earlier stages of the network.
The experiment with eukaryotic stem cell data was conducted with 3997 DHS samples and a kernel length of 11. The convolutional RBM was trained for 300 epochs (every sequence was shown to the model 300 times in the same order), using 25 different kernels.

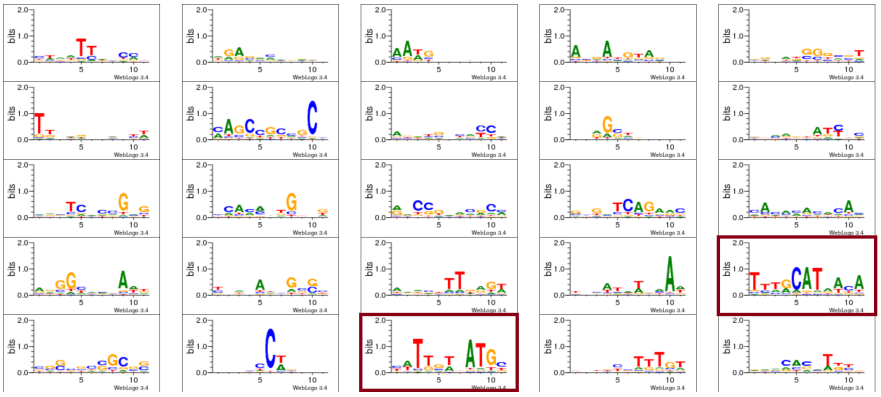

Figure 20: All of the motifs that were learned by the cRBM for the stem cell data. The number of motifs to learn (20 here) and the length of each motif (11 here) are hyper-parameters of the model.

In figure 20, we can see all the motifs extracted by the cRBM. It is especially interesting to note that some motifs seem to focus on very few positions, while the remaining positions have a very low IC. Others appear to be absorbing very general structures in the data set, meaning that a hit from that motif might indicate some general structure of DHSs, such as a high GC-content. Our model only captures patterns in comparison to randomly distributed nucleotides which is why some motifs will always represent patterns that are present in all genomic sequences.
Finally, we see some motifs with high IC that do very likely represent transcription factor binding sites. When we take the 23rd motif from figure 20, we can see that it corresponds to the *oct4* binding site. Figure 21 shows a comparison of the motif found by the cRBM with the one from the JASPAR database. The oct4 motif from the JASPAR database is of length 15, but the most relevant parts of it could be captured by our model.

We plugged the position weight matrices from figure 20 (PWMs) into TOMTOM [20], which is part of the MEME suite. This online tool compares position weight matrices from different databases based on statistical tests. The first hit of the tool is depicted in figure 21 and has an overlap of eleven k-mers as well as a p-value of 2.9×10^{-12}. The p-value is a statistical test to measure how extreme an observation is, given a model (often called *null hypothesis*).
In the case of comparing two motifs, the p-value states the probability that the match of two position weight matrices occurred by chance when the one motif is assumed to generate the observations. A small p-value indicates high similarity between the motifs while a high p-value indicates low similarity.
With a significance level of 0.05 (which is common in literature), the

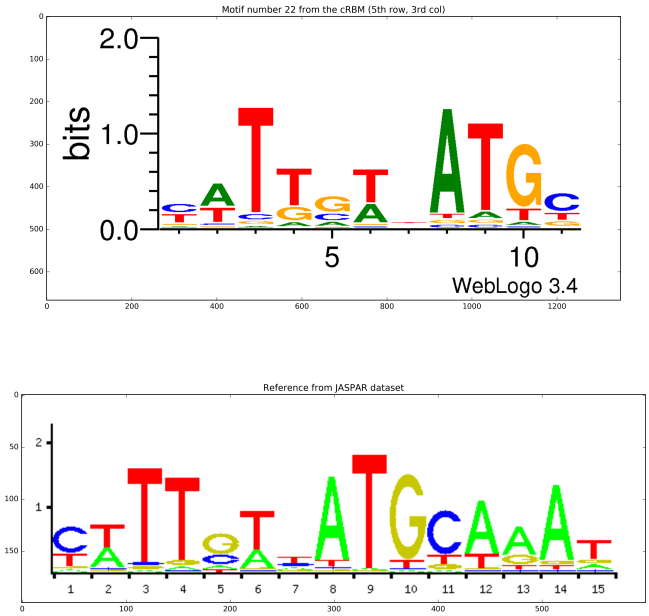

Figure 21: Qualitative comparison of the JASPAR motif for Oct4 and the one found by our model. Note that our motif is only eleven base-pairs long, while the JASPAR motif contains 15 nucleotides.

two motifs are very similar. However, the similarity becomes already evident when considering the web-logos depicted in figure 21.
Furthermore, we also extract the reverse complement of the motif quite well without ever having implemented to search for that in the sequences. The reverse complement was extracted by motif 20 (fourth row, fifth logo) in figure 20 and has a p-value of 6.87×10^{-12}. Due to the difference in length when comparing it to the JASPAR motif, the two motifs highlighted in figure 20 do not look like complements of one another.

The results of the experiment show that the cRBM is capable of detecting meaningful motifs without explicitly looking for it. The evolution of each of the kernels only comes from minimizing the energy of the data.

4.3.2 *Detecting motifs in fibroblast lung cells*

We conducted the same experiment on data from lung fibroblast cells. As with the oct4 data set, the sequences come from the ENCODE database and are freely available. We expect the *mafk* transcription factor to bind to long-fibroblast-specific DHSs and therefore, the *mafk* motif should be over-represented in this data set.

When comparing it to the known web-logo from the JASPAR database, we can see immediately that the web-logos extract more or less the same PWM with the flanking regions of the motif being of different sizes. However, both motifs do not have a high information content in these flanking regions.

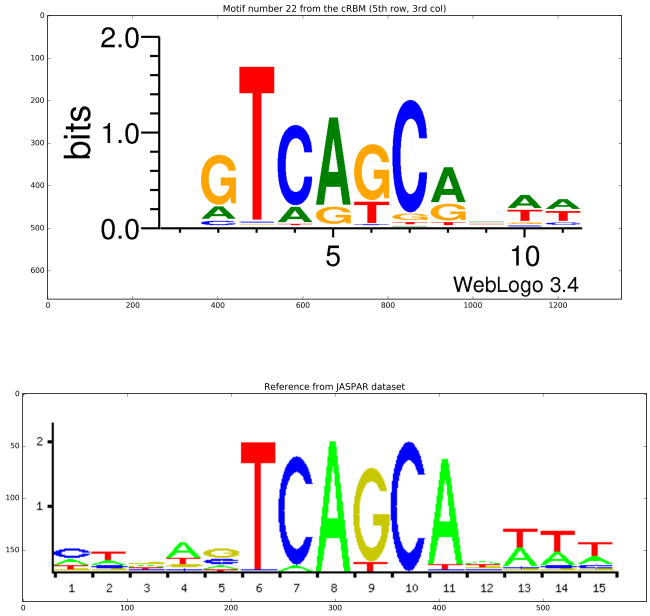

Figure 22: Qualitative comparison of the JASPAR motif for mafk and the one found by our model. Note that our motif is only eleven base-pairs long, while the JASPAR motif contains 15 nucleotides.

When plugging the results from our experiment into TOMTOM, we see that the PWM depicted in figure 22 is very similar to the *mafb* transcription factor binding site. While the first two hits in the JASPAR database are variants of *mafk*, called *maff* & *mafg*, it is almost impossi-

ble to distinguish between the binding sites of the maf-protein family [25]. The *mafk* TFBS has a p-value of 3.27×10^{-7} which is in a similar range to the one for *mafb*. As before with the *oct4* data, the cRBM also extracted the reverse complement of the *mafk* TFBS, listing it already as the second hit in the tool's ranking.

4.4 CLASSIFICATION OF TISSUE SPECIFIC DHS

In the next experiment, I wanted to see if the cRBM can be used for classification and how it compares to other well-known classifiers.
That said, the convolutional RBM is a generative model that computes probability distributions over data and not a classifier. It can, for example, not take into account negative data like a *support vector machine* or neural networks.
Thus, the cRBM is not expected to achieve maximum accuracy when trying to predict which tissue a certain sequence comes from. But still, we wanted to know how well the model learns specific features of the DHSs for a certain tissue type.
In our experiment, we compared the cRBM to a *first order Markov Model* (explained in section 2.1.2) and a *Support Vector Machine* (explained in section 2.1.4). While the Markov Model should symbolize a lower bound (because it only counts frequencies), the SVM is expected to perform better in terms of classification than our model when increasing the size of the data representations.
Furthermore, we compared different cRBM models with different number of motifs to see whether we observe saturation or overfitting of the model.
For the experiment, we once again used the data sets from the previous two experiments, namely the eukaryotic stem cell data and the fibroblast lung cell data.

4.4.1 *Comparing the free energy of a trained model*

The goal was now to train two different cRBMs that would learn on one of the two data sets each. After successful training of both convolutional RBMs, we compared the free energy of both of them. The subtracted free energy is similar to the data log-likelihood-ratio of both models and can be defined as follows:

$$\mathcal{F}(\mathcal{D}) = -\log\left(\sum_{\mathbf{h}} e^{-E(\mathcal{D},\mathbf{h}|\Theta)}\right) \tag{29}$$

The difference between the free energies of both models can be written as:

$$\begin{aligned}\Delta\mathcal{F}(\mathcal{D}) &= \mathcal{F}_{ModelA}(\mathcal{D}) - \mathcal{F}_{ModelB}(\mathcal{D}) && (30)\\ &= -\log\left(\sum_{\mathbf{h}} e^{-E(\mathcal{D},\mathbf{h}|\Theta_1)}\right) + \log\left(\sum_{\mathbf{h}} e^{-E(\mathcal{D},\mathbf{h}|\Theta_2)}\right) && (31)\\ &= \log\left(\frac{\sum_{\mathbf{h}} e^{-E(\mathcal{D},\mathbf{h}|\Theta_2)}}{\sum_{\mathbf{h}} e^{-E(\mathcal{D},\mathbf{h}|\Theta_1)}}\right) && (32)\end{aligned}$$

The data $\mathcal{D}$ contains sequences from both classes that were never shown to any of the two models during training. This test set was now evaluated using the difference of the free energies from equation 32.

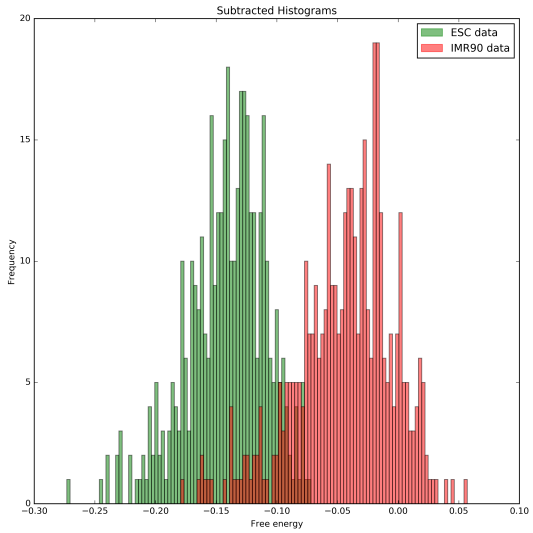

Figure 23: Histogram depicting the difference in free energy for test sequences of both tissue types. We can see that a naive bayesian classifier could yield good classification accuracy.

Figure 23 shows a histogram of the test data for both classes. When we do actual classification depending on whether the difference of the free energies is positive or negative, we can compare the cRBM to other methods.
A ROC curve shows the proportion of true positives to false positives. An optimal classifier would reach the point in the top left corner while complete chance is on the diagonal. Figure 24 compares the cRBM with different importance levels for the sparsity constraint to other models. As expected, the first order markov model is the weakest classifier in figure 24. This might be due to the markov property that only picks up relationships of two successive nucleotides. On the contrary, the support vector machine tries to find the best separation between the two classes by using both, positive and negative examples. The data is represented as 5-mers for the algorithm, such that every input vector is of dimensionality 4^5. This can be interpreted as a vector, containing the number of times every motif of length 5 is present in the data. The SVM should be the upper bound to the capabilities of the cRBM because of the consideration of negative examples and its maximization of the margin.

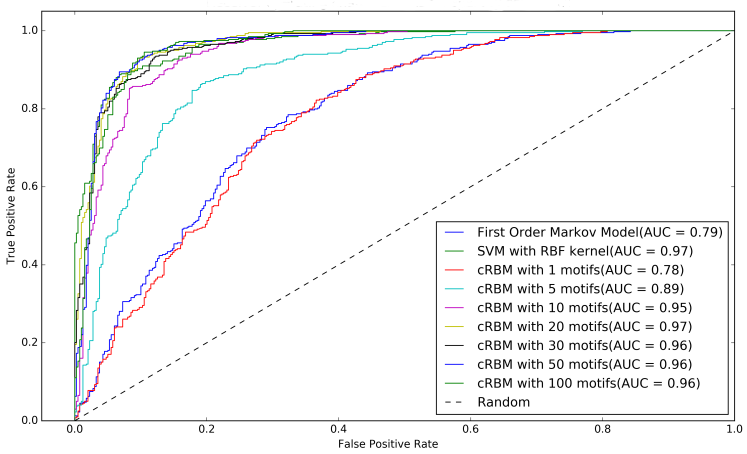

Figure 24: A ROC curve for different trained cRBMs compared to a SVM with gaussian kernel and a first order hidden markov model. While generally more motifs result in better classification, 20-30 motifs seem to be sufficient to capture all of the important specificities for the data set.

However, it seems as if the convolutional RBM can classify almost as well as the SVM with only around 20 motifs. This indicates that the algorithm is learning a very economic representation of the data. The fact that already 20-30 motifs seem to be enough to classify between eukaryotic stem cells and fibroblast lung cells is a clear indicator that the data representation learned is meaningful and also depending strongly on the test set. That means that our model learns non-trivial motifs which could easily be the case due to the over-completeness of the representation compared to the sequence.

4.5 BIAS DETECTION IN PAR-CLIP EXPERIMENTS

So far, the tasks that we could solve with our method were already solved with other techniques or could have been solved with supervised methods, such as deep convolutional neural networks, provided the labels are available.
In this section, we want to analyze PAR-CLIP control experiments and quantify the bias they might introduce. As opposed to the previous two problems where we expected the sequences to contain a high signal, control experiments should contain much more noise and ideally no signal at all. The features learned by the cRBM are expected to reveal the abundance of RNAs in the cell at most.
Any features that we might detect represent a putative bias in the sequencing protocol. Such a bias could be detected and normalized for

by using the likelihood probability of a sequence x given by $Pr(x|\theta)$. In the previous section, we already used the free energy of two cRBM models to discriminate between different tissue types. To detect biases in NGS experiments, we tried a similar approach by training the cRBM on two different data sets. One originates from various PAR-CLIP control experiments and the other comes from a target experiment that was conducted with a particular target protein (*PUM2*) and should have a much higher signal than the unspecific control experiments.
These two models were then compared by using a test set of sequences from both data sets. We evaluated the free energy of both on each of the models, yielding a free energy for background sequences on the background model and vice versa. Evaluating these sequences on the background model gives us a measure of how similar each sequence is to the background. This gives us a measure of how similar a sequence is to the background. A normalization in peak calling could be developed based on such a similarity metric.

Furthermore, the likelihood of the data to belong to the background distribution can be used to see if a target PAR-CLIP experiment is flawed. In such a case, the target distribution would have a mean that is very different from the mean of the background distribution.

The data that the cRBM was trained on came from a fusion of multiple experiments and was not extracted by ourselves. *BackCLIP* is a tool to identify common background in different PAR-CLIP data sets [44] and the authors of it constructed a set of common background which is the intersection between 19 different PAR-CLIP control experiment data sets. Duplicates were removed in the process such that the remaining sequences no longer reflect the abundance of RNA transcripts in the data.
How the data is generated is explained in great detail in the supplementary material of [44] and the sequences are freely available. We use this common background to train the cRBM. Since the sequences did not have the same length and most of the sequences were quite short, we decided to pad them with sequences from the reference genome (hg19). The length distribution of the data is depicted in figure 25. The sequences were preprocessed to have length 100 by using the average between start and endpoint for each line in the bed file and adding resp. subtracting 50 from both. Thus, the sequences all have length 100 and are centered at the region from the bed file.

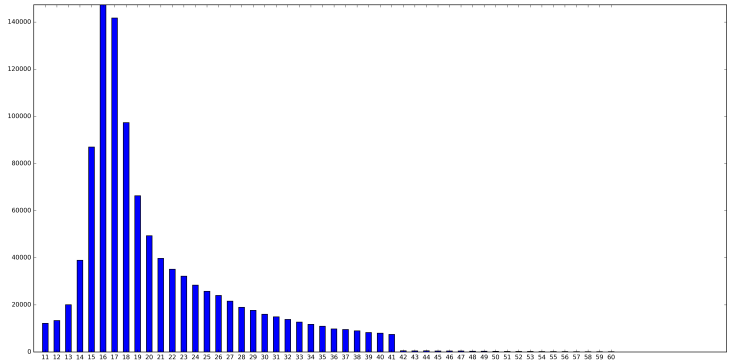

Figure 25: The length distribution of sequences (in nucleotides) in the common background data set. While short reads are more common by far, they are less favorable for our analysis.

4.5.1 *The PAR-CLIP Protocol*

PAR-CLIP is a sequencing technique mainly designed to capture the binding events of RNA-binding proteins (RBPs). It exploits the formation of bonds between RBPs and the RNA when exposed to UV light. For these formations of RNA and protein complex, the RBP of interest is then captured by a specifically designed antibody.
Afterwards, the cell is *lysed* to break down the membrane of the cells and the RBPs of interest are isolated by *immunoprecipitation*. Once the solution has been washed to remove any non-specific RNA, only the ensemble of RNA and RBPs of interest remain.
Adapters are now added to both ends of the RNA for various technical reasons. The adapters are needed for amplification of the RNAs of interest but can also be used to "barcode" the samples with an artificial sequence added to both ends.
Finally, the RBP is digested and sequencing takes place. The result is then a set of transcripts, usually represented as multiple strings in a text file. Some of the steps of the PAR-CLIP protocol have not been addressed here as they would exceed the limitations of the thesis.

It has been shown that biases in PAR-CLIP sequencing are present and that control experiments with unspecific antibodies help to circumvent the problem [3]. In the usual setup, a control experiment is conducted to complement each experiment with target specific antibodies.
There are several ways to introduce a systematic bias to the protocol, and random errors can occur as well. While the latter is less problematic and easier to compensate by applying probabilistic algorithms in the downstream analysis, systematic biases give rise to serious prob-

lems [15].
Friedersdorf et al. have shown that background binding is both, common and systematic in PAR-CLIP experiments [17]. It results mainly from the inefficiency of the UV light cross-linking procedure which supports the formation of covalent bonds between RNA and protein. Furthermore, all NGS techniques contain systematic biases as, for instance, the favorisation of GC-rich content. [5]

Our goal is to learn a probability distribution over the background and use it to examine sequences from target experiments. If the likelihood of a sequence to come from the background distribution is high, we might consider it a falsely sequenced transcript.

4.5.2 *Identifying Motifs In Control Experiments*

We trained our model on the data set described in the previous paragraph. It is not expected that the model learns strong motifs for two reasons. First, we have moved from the DNA domain to RNAs. RNA binding proteins are shown to have a binding preference to a certain sequence but also to the structure that the RNA forms in space [33]. Thus, the sequences do not show patterns as strongly as they would in a DNA setting.
Furthermore, we are analyzing background which per definition has less prominent features in it than target experiments. The motifs

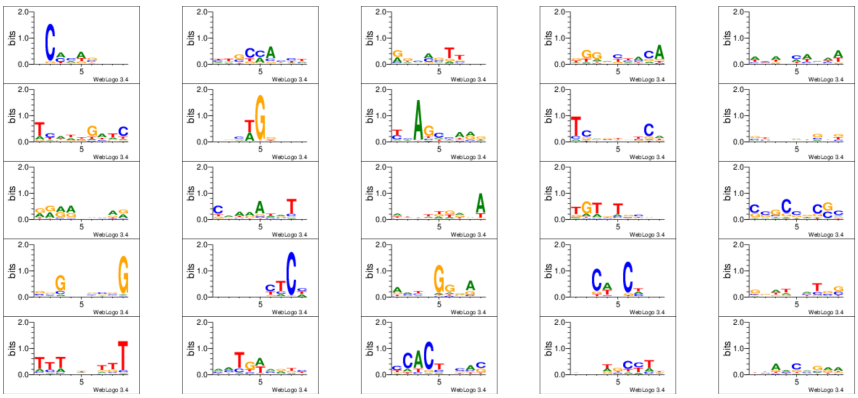

Figure 26: Motifs learned by the cRBM. While the motifs are much less prominent than the DHS data from chapter 4.3.1, the model still finds over-represented motifs.

learned by the cRBM are depicted in figure 26. We can observe clearly that there is a higher GC-content in the learned web-logos and that otherwise, the motifs are not clearly identifiable. As with the trials before, we used TOMTOM to check if any of the found motifs are known in literature.

The cRBM detected a binding motif for the *HNNRPLL* protein coding gene with a p-value < 0.01 (figure 26, third row, fourth column).
HNNRPLL is known to be a master regulator for alternative splicing and supports the stability of mRNAs [41]. This is why it makes sense that the protein binds RNA unspecifically and thus appears over-represented in the dataset.
Similarly, we detect a binding site for the *RBM4* protein in the third row and fifth column with a p-value of less than 0.01 and the *HuR* protein with also a p-value of less than 0.01.
HuR is also described to be involved in the stability of mRNA and should thus be present in background binding [1].
However, many of the other motifs do not produce any matches. They might have learned only noise from the data which also is logical because a good representation of noisy data will also try to capture it to maximize the posterior probability.

4.5.3 *Discrimination Between Control And Target*

Next, we performed an analysis of the mean free energy (equation 29) between background and target experiments. For that, we trained a cRBM model on *PUM2* target RBP and compared it to the model from the previous section (section 4.5.2).

On test sets that were held out during training of both models, we evaluated and compared them. It is worth noting that the test set was never used before, also not for evaluating the test error of the cRBMs which could have an indirect effect on a model because stopping criteria would have been chosen according to it. Therefore, it is considered best practice not to use the test set at all prior to the evaluation [13].
For both test sets, we evaluated the free energy on both models and plotted a colored histogram over the sequences. Figure 27 shows the mean free energy of test sequences colored in red, green and blue. All sequences were evaluated on the background model which means the energy is related to the probability of a sequence to come from the background distribution. While the absolute values are rather hard to interpret, the relative relationships between each of the three datasets are interesting.

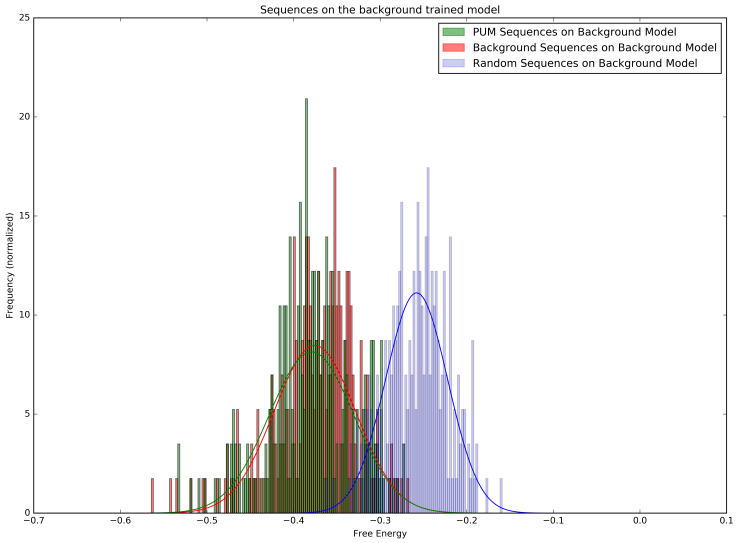

Figure 27: Evaluation of the free energy of background and *PUM2* test sequences. Evaluated on the background model, both sets are very similar in terms of their free energy. The random sequences, however, are much less probable.

In red are the background sequences while the sequences from the target experiment are colored in green. To make both values more comparable, we further added a set of random sequences.

As expected, the random sequences are less probable to belong to the background distribution than the other two. However, it seems as if sequences from the *PUM2* target experiment have approximately the same likelihood to belong to the background as the background sequences themselves.

This might be reasonable because the background distribution is expected to be much broader than the target distribution. However, classifying between target and control sequences based on the evaluation on the background model will not be accurate.

Respectively, we also did the complementary analysis in which we compared all three test sets (background, target and random) on the model trained on target sequences. The results from the comparison are depicted in figure 28.
In this case, the test set of *PUM2* sequences was the most probable one while the background sequences had a higher mean free energy. It is interesting to see, however, that the background distribution is still reaching far into the target distribution while the overlap between the target and the random distribution is only marginal.

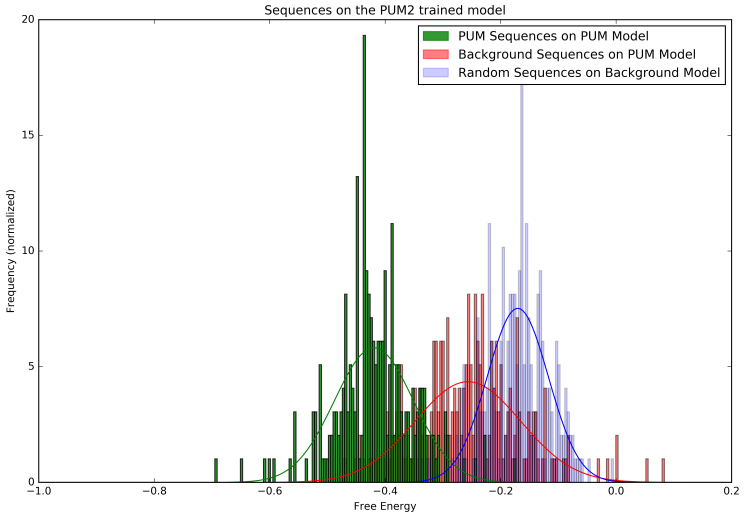

Figure 28: Evaluation of the free energy of background and *PUM2* test sequences. Evaluated on the target model, the sequences from the *PUM2* experiment are much more probable than the background sequences. The random sequences show the least similarity to the target model.

We see that target and control sequences seem to be very similar to the model trained on control sequences and distinct to the model trained on target sequences. Therefore, the background distribution is much broader which was expected.

5

CONCLUSION & OUTLOOK

The goal of this thesis was to investigate if the convolutional restricted Boltzmann Machine can be used to learn good representations of genomic sequences and if its unsupervised nature can be used to detect biases in sequencing protocols.
The model was implemented together with several extensions to the original algorithm and several tests have shown that meaningful features are learned.
Furthermore, a cRBM can be trained on sequencing data from PAR-CLIP control experiments. Using the data likelihood, we can distinguish between control and target experiments. From here, it might be possible to develop a tool which can reliably determine if a particular sequence originates from an unintended binding event or whether the sequence was pulled out correctly by the antibody.

From the conducted tests, it is clear that cRBMs can be used efficiently to find good data representations of genomic sequences.

5.1 SUMMARY

I presented the convolutional restricted Boltzmann machine as an algorithm to learn an efficient representation of sequences. Other recent approaches have proven that deep learning algorithms can be used to capture patterns in NGS data. CRBMs are an unsupervised building block of these deep learning techniques but have not yet been applied to biology.
But while neural networks have the reputation of being hard to interpret, the features learned by their convolutional counterpart are easy to understand.
In the application of cRBMs to biological sequences, the filters correspond to motif detectors, thus allowing for very natural interpretation. The unsupervised character of the model allows it to be applied to problems where labels are hard to get and is one of the points

where the method differs from cNN architectures that were recently applied to genomic sequences.

I have introduced the mathematical foundations that underlay our implementation together with most of the extensions to the original RBM training procedure. It was further shown that the model can learn meaningful representations of DHS sequences and that by doing so it recovers known motifs from literature using the sequences alone.
Additionally, these representations are specific to a tissue type and strongly depend on the data presented to the model. This property allows for accurate classification between data from different tissue types and can also be regarded as a proof-of-concept for the analysis of background in NGS protocols.
Tests on background data from PAR-CLIP experiments show that the data likelihood function of cRBMs can be used to determine if a sequence belongs to the background or not. The detection of such biases in an unsupervised fashion can improve peak-calling and thus help downstream analysis of experiments.
While the model is already working as expected, such a tool still requires work and improvements upon the existing implementation which are intended to be done in the future.

5.2 OUTLOOK

So far, we have only been looking at one-layered cRBMs. In the future, it would be interesting to see how the representation of sequences can be improved in a deep learning scenario as depicted in figure 29.
In such a setting, multiple cRBMs are stacked on top of each other, making the learning of hierarchical features possible. The second hidden layer would learn motifs that are made up from motifs from the first layer. Such a network is called a *deep belief network* (DBN) and has been shown to learn meaningful high-level features in computer vision applications [32].
Another way to stack cRBMs could be to use a standard RBM model for the second layer, thus capturing the interactions between motifs.
Both of these stacking approaches have clear biological interpretations and can be visualized well (one as motif logos by projecting motifs from higher layers back to input space, the other as a graph connecting motifs with each other).
Deep architectures have been shown to add much more representative power to a model because units in higher layers are able to grasp global structures in the data while the first layer features describe prominent patterns on a local level.

Another interesting aspect would be to develop a method that performs peak-calling without the need for an explicit control experiment but rather an ensemble from previous controls.

A cRBM would be trained on these different control datasets and sequences coming from new target PAR-CLIP experiments could be evaluated on that model. The free energy (or data likelihood, if the partition function can be estimated) would then tell whether the experiment went well and also which sequences belong to the background.
Such a method would reduce the cost of PAR-CLIP experiments significantly because the need to conduct control experiments would be eliminated.

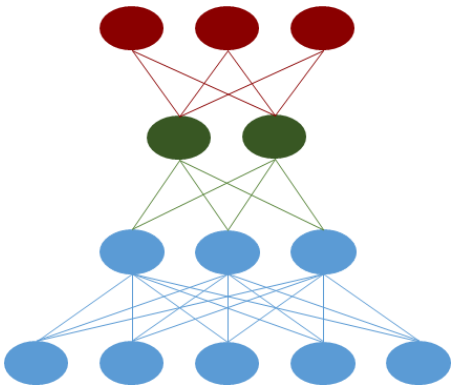

Figure 29: Schematic view of a deep belief network

Many smaller improvements to the existing implementation will be incorporated in the future, such as an estimation of the data likelihood probability instead of measuring only the free energy. This is especially necessary when the model is not used in a comparative way where the ratio of free energy does not require the computation of the partition function.

It is a severe constraint on the model that it can only use one single graphics card for training. While it is possible that this issue will be resolved as the Theano framework evolves, an effort can be made to process batches of the training on different GPUs and speed up the training procedure.

BIBLIOGRAPHY

[1] Kotb Abdelmohsen et al. "Posttranscriptional orchestration of an anti-apoptotic program by HuR". In: *Cell cycle* 6.11 (2007), pp. 1288–1292.

[2] Babak Alipanahi et al. "Predicting the sequence specificities of DNA-and RNA-binding proteins by deep learning". In: *Nature biotechnology* (2015).

[3] Manuel Ascano et al. "Identification of RNA–protein interaction networks using PAR-CLIP". In: *Wiley Interdisciplinary Reviews: Rna* 3.2 (2012), pp. 159–177.

[4] Timothy L Bailey et al. "MEME SUITE: tools for motif discovery and searching". In: *Nucleic acids research* (2009), gkp335.

[5] Yuval Benjamini and Terence P Speed. "Summarizing and correcting the GC content bias in high-throughput sequencing". In: *Nucleic acids research* (2012), gks001.

[6] James Bergstra et al. "Theano: a CPU and GPU Math Expression Compiler". In: *Proceedings of the Python for Scientific Computing Conference (SciPy)*. Oral Presentation. Austin, TX, June 2010.

[7] C.M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, 2006. ISBN: 9780387310732. URL: https://books.google.de/books?id=kTNoQgAACAAJ.

[8] Sharan Chetlur et al. "cudnn: Efficient primitives for deep learning". In: *arXiv preprint arXiv:1410.0759* (2014).

[9] Peter JA Cock et al. "Biopython: freely available Python tools for computational molecular biology and bioinformatics". In: *Bioinformatics* 25.11 (2009), pp. 1422–1423.

[10] Ronan Collobert and Jason Weston. "A unified architecture for natural language processing: Deep neural networks with multitask learning". In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 160–167.

[11] ENCODE Project Consortium et al. "An integrated encyclopedia of DNA elements in the human genome". In: *Nature* 489.7414 (2012), pp. 57–74.

[12] Gavin E Crooks et al. "WebLogo: a sequence logo generator". In: *Genome research* 14.6 (2004), pp. 1188–1190.

[13] Scikit-learn developers. *Cross-validation: evaluating estimator performance*. `http://scikit-learn.org/stable/modules/cross_validation.html`. Accessed: 2016-07-22. 2014.

[14] Patrik D'haeseleer. "What are DNA sequence motifs?" In: *Nature biotechnology* 24.4 (2006), pp. 423–425.

[15] Aaron Diaz et al. "Normalization, bias correction, and peak calling for ChIP-seq". In: *Stat Appl Genet Mol Biol* 11.3 (2012), p. 9.

[16] Asja Fischer and Christian Igel. "An introduction to restricted Boltzmann machines". In: *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*. Springer, 2012, pp. 14–36.

[17] Matthew B Friedersdorf and Jack D Keene. "Advancing the functional utility of PAR-CLIP by quantifying background binding to mRNAs and lncRNAs". In: *Genome biology* 15.1 (2014), pp. 1–16.

[18] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*. Vol. 1. Springer series in statistics Springer, Berlin, 2001.

[19] David S Gross and William T Garrard. "Nuclease hypersensitive sites in chromatin". In: *Annual review of biochemistry* 57.1 (1988), pp. 159–197.

[20] Shobhit Gupta et al. "Quantifying similarity between motifs". In: *Genome biology* 8.2 (2007), p. 1.

[21] Geoffrey Hinton. "A practical guide to training restricted Boltzmann machines". In: *Momentum* 9.1 (2010), p. 926.

[22] Geoffrey E Hinton. *Neural Networks for Machine Learning*. `https://www.coursera.org/learn/neural-networks`. Accessed: 2016-07-22. 2014.

[23] Geoffrey E Hinton. "Training products of experts by minimizing contrastive divergence". In: *Neural computation* 14.8 (2002), pp. 1771–1800.

[24] Geoffrey E Hinton and Ruslan R Salakhutdinov. "Reducing the dimensionality of data with neural networks". In: *Science* 313.5786 (2006), pp. 504–507.

[25] Meenakshi B Kannan, Vera Solovieva, and Volker Blank. "The small MAF transcription factors MAFF, MAFG and MAFK: current knowledge and perspectives". In: *Biochimica et Biophysica Acta (BBA)-Molecular Cell Research* 1823.10 (2012), pp. 1841–1846.

[26] David R Kelley, Jasper Snoek, and John L Rinn. "Basset: Learning the regulatory code of the accessible genome with deep convolutional neural networks". In: *Genome research* (2016).

[27] Hugo Larochelle and Yoshua Bengio. "Classification using discriminative restricted Boltzmann machines". In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 536–543.

[28] Yann LeCun. *L'apprentissage profond: une révolution en intelligence artificielle*. http://www.college-de-france.fr/site/yann-lecun/course-2016-02-26-11h00.htm. Accessed: 2016-06-26. 2015.

[29] Yann A LeCun et al. "Efficient backprop". In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.

[30] Yann LeCun, Corinna Cortes, and Christopher JC Burges. *The MNIST database of handwritten digits*. 1998.

[31] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[32] Honglak Lee et al. "Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations". In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML '09. New York, NY, USA: ACM, 2009, pp. 609–616. ISBN: 978-1-60558-516-1. DOI: 10.1145/1553374.1553453. URL: http://doi.acm.org/10.1145/1553374.1553453.

[33] Xiao Li et al. "Predicting in vivo binding sites of RNA-binding proteins using mRNA secondary structure". In: *Rna* 16.6 (2010), pp. 1096–1107.

[34] Andrei Lihu and Ştefan Holban. "A review of ensemble methods for de novo motif discovery in ChIP-Seq data". In: *Briefings in bioinformatics* (2015), bbv022.

[35] Jonas Maaskola and Nikolaus Rajewsky. "Binding site discovery from nucleic acid sequences by discriminative learning of hidden Markov models". In: *Nucleic acids research* 42.21 (2014), pp. 12995–13011.

[36] Philip Machanick and Timothy L Bailey. "MEME-ChIP: motif analysis of large DNA datasets". In: *Bioinformatics* 27.12 (2011), pp. 1696–1697.

[37] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[38] Anthony Mathelier et al. "JASPAR 2016: a major expansion and update of the open-access database of transcription factor binding profiles". In: *Nucleic acids research* (2015), gkv1176.

[39] Matthew Mayo. *Top 5 arXiv Deep Learning Papers, Explained*. http://www.kdnuggets.com/2015/10/top-arxiv-deep-learning-papers-explained.html. Accessed: 2016-07-21. 2015.

[40] Mohammad Norouzi. *Convolutional Restricted Boltzmann Machines for Feature Learning*. 2009.

[41] Shalini Oberdoerffer et al. "Regulation of CD45 alternative splicing by heterogeneous ribonucleoprotein, hnRNPLL". In: *Science* 321.5889 (2008), pp. 686–691.

[42] Pavel A Pevzner, Sing-Hoi Sze, et al. "Combinatorial approaches to finding subtle signals in DNA sequences." In: *ISMB*. Vol. 8. 2000, pp. 269–278.

[43] Naim U Rashid et al. "ZINBA integrates local covariates with DNA-seq data to identify broad and narrow regions of enrichment, even within amplified genomic regions". In: *Genome biology* 12.7 (2011), p. 1.

[44] PH Reyes-Herrera et al. "BackCLIP: a tool to identify common background presence in PAR-CLIP datasets". In: *Bioinformatics* (2015), btv442.

[45] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[46] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. `http://sebastianruder.com/optimizing-gradient-descent/index.html`. Accessed: 2016-06-02.

[47] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning internal representations by error propagation*. Tech. rep. DTIC Document, 1985.

[48] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[49] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. "Restricted Boltzmann machines for collaborative filtering". In: *Proceedings of the 24th international conference on Machine learning*. ACM. 2007, pp. 791–798.

[50] Pierre Sermanet and Yann LeCun. "Traffic sign recognition with multi-scale convolutional networks". In: *Neural Networks (IJCNN), The 2011 International Joint Conference on*. IEEE. 2011, pp. 2809–2813.

[51] Johannes Stallkamp et al. "The German traffic sign recognition benchmark: a multi-class classification competition". In: *Neural Networks (IJCNN), The 2011 International Joint Conference on*. IEEE. 2011, pp. 1453–1460.

[52] Robert E Thurman et al. "The accessible chromatin landscape of the human genome". In: *Nature* 489.7414 (2012), pp. 75–82.

[53] Tijmen Tieleman. "Training restricted Boltzmann machines using approximations to the likelihood gradient". In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 1064–1071.

[54] Philip J Uren et al. "Site identification in high-throughput RNA–protein interaction data". In: *Bioinformatics* 28.23 (2012), pp. 3013–3020.

[55] Ya-Mei Wang et al. "Correlation between DNase I hypersensitive site distribution and gene expression in HeLa S3 cells". In: *PloS one* 7.8 (2012), e42414.

[56] Paul Werbos. "Beyond regression: New tools for prediction and analysis in the behavioral sciences". In: (1974).

[57] D Randall Wilson and Tony R Martinez. "The general inefficiency of batch training for gradient descent learning". In: *Neural Networks* 16.10 (2003), pp. 1429–1451.

[58] Jeffrey Wood and John Shawe-Taylor. "Representation theory and invariant neural networks". In: *Discrete applied mathematics* 69.1 (1996), pp. 33–60.

[59] Carl Wu et al. "The chromatin structure of specific genes: I. Evidence for higher order domains of defined DNA sequence". In: *Cell* 16.4 (1979), pp. 797–806.

[60] Simon X Yang and Max Meng. "An efficient neural network approach to dynamic robot motion planning". In: *Neural Networks* 13.2 (2000), pp. 143–148.

[61] Sai Zhang et al. "A deep learning framework for modeling structural features of RNA-binding protein targets". In: *Nucleic acids research* (2015), gkv1025.

[62] Bolei Zhou et al. "Learning deep features for scene recognition using places database". In: *Advances in neural information processing systems*. 2014, pp. 487–495.

LIST OF FIGURES