Universiteit Leiden
Opleiding Informatica

Combining graph mining and deep learning

in molecular activity prediction

Name: Hanjo Boekhout

Date: 19/08/2015

1st supervisor: Siegfried Nijssen

2nd supervisor: Aske Plaat

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)

Leiden University

Niels Bohrweg 1

2333 CA Leiden

The Netherlands

Combining graph mining and deep learning in molecular activity prediction

Hanjo Boekhout

Abstract

During drug discovery an important activity is the prediction of molecular activity. We want to discover if combining the graph mining method gSpan with the deep learning classifier Deep Belief Networks (DBN) provides accurate prediction. gSpan will be used to identify frequent substructures. By representing molecules as a set of these substructures we translate a molecular data set into numerical data sets which can be used for classification (prediction). We extend gSpan with two search methods, multiple sequential coverage and tree search, because we want to know what set of substructures provides the best representation of molecules for classification with DBNs. Our experiments will show that the substructures provided by tree search lead to the best classification results with DBNs. To determine whether a DBN is better at classifying such a data set than other classifiers, we compare uniform DBNs with several WEKA classifiers. Experiments will show that uniform DBNs, which are pre-trained, predict with less accuracy than multi-layer perceptrons, but classify with similar and sometimes better accuracy than some tree, boosting and SVM based classifiers in WEKA.

Acknowledgements

I would like to thank K. Putman for allowing me to use his server to run some of my experiments.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

With so many diseases in the world that we can not cure yet, drug discovery is one of the most important activities in the field of medicine. The process of drug discovery is however long, expensive and prone to failure. During drug discovery it is often known which properties a new drug should have in order to be effective. Most failures in drug discovery can be attributed to drugs not having such a property, while it was expected that the drug would have that property. This makes molecular activity prediction, predicting the property of a molecule, a key aspect as it allows a failure to be caught early, saving a lot of time and money. Making such predictions is however far from trivial. Usually activity prediction is performed through the identification of a certain molecular property or substructure from a set of molecules that are known to have (or not have) the property we are trying to predict. Over the years the volume of data, concerning molecules and their properties, has increased and as such identifying important substructures has become a data mining problem.

One of the main interests in the field of graph mining is mining graph data sets for graphs that are subgraphs of a large number of instances. Between the mid-1990s and mid-2000s the need for fast and efficient algorithms to perform this task lead to the development of a range of algorithms like MoFa [BB02], FFSM [HWP03], gSpan [YH02] and Gaston [NK04]. It is common to represent the two-dimensional structure of molecules as graphs, as shown in Figure 1.1. The labeled nodes of these graphs represent the atoms of the molecule and the labeled edges represent the bonds between the atoms. Applying graph mining to molecular databases now becomes interesting for drug discovery as we can search for those subgraphs (substructures) that are frequent in one part of the database but infrequent in the other. These substructures divide the database into active and inactive molecules [BB02] and therefore have the potential for predicting the property of a new drug or being used as a base in the development of a new drug. Because not all subgraphs found through graph mining are well suited for prediction, it is key to find those substructures that are relevant for biological or physicochemical properties.

In molecular activity prediction it is common for experts to determine relevant substructures instead of automatically generating them. Although this guarantees your results to be relevant biologically, it limits the set of relevant substructures to the knowledge of the expert. This means that new relevant substructures would be hard to find. Because automatically finding substructures through graph mining is not restricted by the knowledge of an expert, we are able to discover new relevant substructures. Automatically generating the set of substructures is also likely to be faster than an expert.

As in [KNK+06], combining the graph mining algorithms with data mining can lead to the discovery

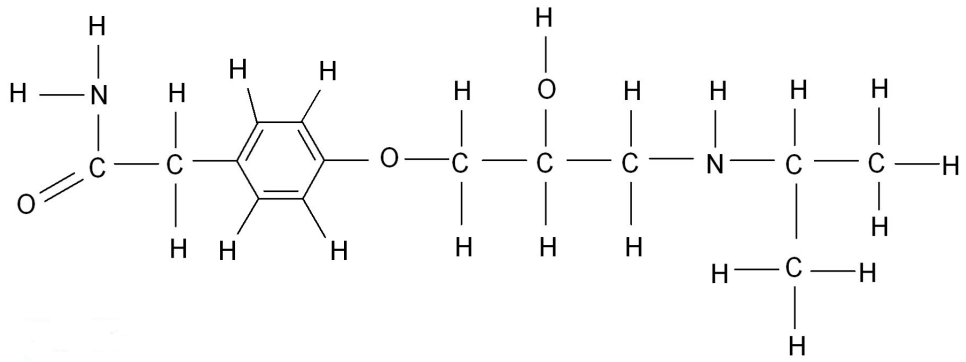

Figure 1.1: The two-dimensional represetation of a molecule as a graph.

of substructures that are statistically relevant for biological or physicochemical properties. Although there has been much research into the synthesis of substructures, such as graph mining, much less attention has been placed on the data mining methods used in classification. It is of interest to find a classifier that is both accurate and leads to the discovery of new relevant substructures. In this thesis we will combine the graph mining algorithm gSpan with deep learning to determine its potential as a classifier in this area.

We have chosen to use a deep learning classifier for this task because of its rising popularity in pattern recognition in recent years. Deep learning has been shown to work well in pattern recognition related areas, such as speech recognition [HDY+12], image analysis [LGRN09] and natural language understanding [SHD14]. The classification of molecules can be considered as determining a set of substructures that determine its class. Therefore recognizing such sets (patterns) can lead to accurate classification. It is because of this aspect of pattern recognition in the classification of molecules that we expect a deep learning classifier to do well. We also expect that relevant combinations of substructures could be found through the use of deep learning. This is of interest because usually only the relevance of a single substructure is considered. We will however not analyze which combinations are found during the experiments in this thesis.

1.1 Thesis Overview

The remainder of this thesis is organized as follows. In chapter 2 we will define the concepts of graph mining and deep learning and explain the gSpan algorithm as well as the Deep Belief Network classifier. In Chapter 3 we discuss related work. Chapter 4 discusses some modifications to gSpan we made and explain how we combined gSpan with the Deep Belief Network. In chapter 5 we elaborate our experiments and analyze their results. Finally chapter 6 contains our conclusions.

Chapter 2

Definitions

In this chapter we will discuss the concepts of the graph mining and deep learning problems. For each of these concepts we will introduce an approach to solve them.

2.1 Graph Mining

Graph mining, or more accurately frequent subgraph mining, is the process of discovering frequent subgraphs from a set of graphs. In order to describe this more formally, we first need the definition of a graph, subgraph isomorphism and support.

Definition 2.1. *A **labeled graph** is a 4-tuple, $G = (V, E, L, l)$, where*

V is a set of vertices,
$E \subseteq V \times V$ is a set of edges,
L is a set of labels,
l: $V \cup E \rightarrow L$, l is a function assigning labels to the vertices and edges.

Because we will be representing molecules as labeled graphs, we will not consider unlabeled graphs.

Definition 2.2. *Given a graph G, a graph G' is said to be **isomorphic** to G if there exists a bijective function f: $V(G) \rightarrow V(G')$, such that*

$\forall u \in V(G)$, $l_G(u) = l_{G'}(f(u))$, and
$\forall (u, v) \in E(G)$, $(f(u), f(v)) \in E(G')$ and $l_G(u, v) = l_{G'}(f(u), f(v))$.

*G is **subgraph isomorphic** to G' if there is an isomorphism from G to a subgraph of G'.*

Definition 2.3. *Given a set of graphs, $GS = \{G_i | i = 0 \dots n\}$, let*

$$\varsigma(g, G) = \begin{cases} 1 & \text{if } g \text{ is isomorphic to a subgraph of } G, \\ 0 & \text{if } g \text{ is not isomorphic to any subgraph of } G \end{cases}$$

*The **support** of g in GS is given by*

$$support(g, GS) = \sum_{G_i \in GS} \varsigma(g, G_i)$$

We can now define frequent subgraph mining.

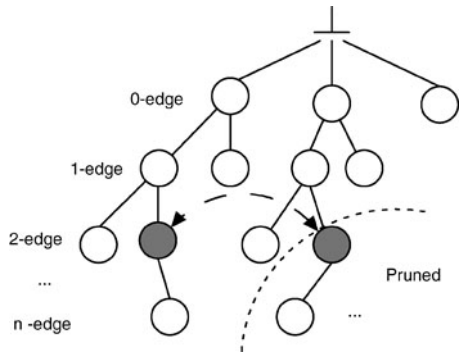

Figure 2.1: DFS Code Tree [YH02]

Definition 2.4. *Given a set of graphs, $GS = \{G_i | i = 0 \dots n\}$, and a minimum support, minSup, let*

$F = \{g | support(g, GS) \geq minSup\}$

*The process of finding F is called **frequent subgraph mining**.*

We have chosen to use the gSpan algorithm to perform this task.

2.1.1 gSpan

gSpan, introduced in [YH02], is a subgraph mining algorithm that adopts depth-first search (DFS). To support DFS, a DFS lexicographical order was developed, which assigns a unique minimum DFS code to each graph. Based on these DFS codes, the search space can be constructed in the form of a DFS code tree as depicted in Figure 2.1.

We see that we start our search with 0-edge graphs, graphs consisting of only a single vertex. From there, we continue the search by adding a single edge at every level. The same graph can however show up in multiple branches. The DFS lexicographical order ensures that the first occurence of a graph in a DFS walk in the search space will equal the minimal DFS code of that graph. This allows us to prune whenever there is a node whose DFS code does not match the minimal DFS code for the graph it represents. Assuming that the dark nodes in Figure 2.1 represent the same graph, we can then prune the entire subtree of the right node. Because every supergraph of an infrequent graph will also be infrequent, we can prune the subtree of any infrequent graph.

Subprocedure 1 is the pseudo-code for a single step, the expansion of one node, in the DFS. Line 1 prevents the expansion of any duplicate node and line 6 ensures we only expand frequent graphs.

Subprocedure 1 Subgraph_Mining(GS, S, s).

```
1: if s ≠ min(s)
2:     return;
3: S ← S ∪ {s};
4: generate all potential children of s in the DFS tree with one edge growth;
5: Enumerate(s); (determine, for each child of s, its support in {g|g ∈ s.GS})
6: for each c, c is a child of s, do
7:     if support(c) ≥ minSup
8:         s ← c;
9:         Subgraph_Mining(GS, S, s);
```

The function Enumerate(s), on line 5, is used to determine the support for each potential child of s. Because each of these children is formed by adding one edge to s and such graphs can only be subgraphs of graphs of which s is a subgraph, we only need to check if they occur in graphs in $s.GS$. $s.GS$ is the set of molecules in which s occurs and is unique for each s. Having determined the support of each child of s, line 7 can prune any potential children that are infrequent.

The pseudo-code of gSpan is outlined in Algorithm 1, where GS is the graph dataset and S is the result set.

Algorithm 1 GraphSet_Projection(GS,S).

1: sort the labels in GS by their frequency;
2: remove infrequent vertices and edges;
3: relabel the remaining vertices and edges in descending frequency;
4: $S^1 \leftarrow$ all frequent 1-edge graphs in GS;
5: sort S^1 in DFS lexicographic order;
6: $S \leftarrow S^1$;
7: **for each** edge $e \in S^1$ **do**
8: initialize s with e, set $s.GS = \{g | \forall g \in GS, e \in E(g)\}$;
9: Subgraph_Mining(GS, S, s);
10: $GS \leftarrow GS - e$;
11: **if** $|GS| < minSup$
12: **break**;

The algorithm starts by removing any infrequent vertices and edges and relabeling the remaining in descending frequency. It then constructs all frequent 1-edge graphs, sorts them in DFS lexicographic order and adds them to the result set. Lines 7-12 continuously perform a DFS on one of the 1-edge graphs and removes them from the graph dataset. We are allowed to remove the edge from the graph dataset, because the DFS already found all frequent subgraphs containing that edge. Executing this algorithm will thus lead to the discovery of all frequent subgraphs.

2.1.2 Implementation

In our experiments we used an implementation of gSpan provided by S. Nijssen, which was used in [BZRN06]. In this implementation some changes were made to the base gSpan algorithm to facilitate a different goal. Unlike the original gSpan algorithm, we do not wish to find frequent subgraphs, but we want to find the subgraphs that best subdivide the dataset between two classes. In other words, preferring frequent subgraphs that are mostly of one class over frequent subgraphs that are well distributed. Running the program requires specifying the number of subgraphs to find. If, for example, we want to find the 100 best subgraphs, we can use the score of the 100th best found subgraph to prune.

The multi-class χ^2 correlation measure, as discussed in [NK05], is used to determine how well a subgraph divides the dataset. However this means that we are no longer interested in the support of a subgraph and thus lose the minimum support threshold as a source of pruning. To prune with the minimum support threshold, we used the fact that any child in the DFS tree would have a smaller or equal support to that of its parent. If this child would, for example, only lose support in one of the classes, the χ^2 correlation measure would assign a higher value to this child than to its parent. Therefore we cannot directly use this measure to prune. [NK05] showed how the χ^2 correlation measure could still be used to perform pruning by using a set of frequency thresholds.

2.2 Deep learning

In deep learning we train deep networks to become a strong classifier for some classification problem. To understand what a deep network is and how it works to classify a given problem we define a network and an activation function.

Definition 2.5. *A **network** is a set of nodes that are grouped in layers. A network consists of an input layer (I), some amount of hidden layers (H_i) and an output layer (O). Let*

$L =$ *amount of layers,*
$N(i) =$ *nodes in layer i,*
$\boldsymbol{n}_i =$*vector of values in nodes of layer i*
$N = \{N(I), N(H_0), N(H_1), \ldots, N(O)\}$*, the entire network.*

A network always start with the input layer and ends with the output layer. There can only exist connections between nodes from subsequent layers. Let

$W_i =$ *the weight matrix for connections between nodes from layer i to* $i+1$
$W_i(j,k) =$ *the weight of the connection between node j in layer i to node k in layer* $i+1$
$\boldsymbol{b}_i =$ *the bias vector that is used as input for layer i*

A weight of zero indicates there is no connection. All connections are directed.

Definition 2.6. *Given the vector $\boldsymbol{n}_i$, weights matrix W_i and the bias vector $\boldsymbol{b}_{i+1}$ the **activation function** for layer $N(i+1)$ is given by*

$$\boldsymbol{n}_{i+1} = \sigma(\boldsymbol{b}_{i+1} + W_i \boldsymbol{n}_i),$$

with

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

Definition 2.7. *A **deep network** is a network that contains multiple hidden layers.*

Definition 2.8. *A **feedforward network**, is a network without connections to previous layers, i.e. all connections are directed from the input layer towards the output layer.*

Let us assume that there is some classification problem, where each instance has n attributes and we want to classify each instance to one of m distinct classes. We can define a network to have $|N(I)| = n$ and $|N(O)| = m$ and choose appropiate weights. We can then provide the attributes of an instance as input for the network ($\mathbf{n}_I$) and use the activation function to calculate the remaining layers of the network. The output layer of the network would then represents the classification of the instance by the network. Because each node represents one class, the classification is given by the node in the output layer with the highest activation. We can thus check if the network correctly classified the instance.

Definition 2.9. *Given a network N and a dataset D, let*

$$e_i = \begin{cases} 0 & \text{if } N \text{ correctly classifies instance } i, \text{ with } i \in D, \\ 1 & \text{otherwise.} \end{cases}$$

*The **error rate** is then given by*

$$e = \frac{\sum_{i \in D} e_i}{|D|}$$

Changing the weights of a network in an attempt to improve classification, i.e. reduce the error rate, is called training the network.

Definition 2.10. *In training a network, we refer to an **epoch** as using every instance of the training dataset once to train the network. Thus, 100 epochs refers to using every instance 100 times.*
*The **learning rate** determines how much of an error, during training, is propagated onto the weights.*

The shape of the network and the method of training differs for each deep learning method. Each method will therefore lead to a different error rate, which we hope to minimize. We have chosen to use a Deep Belief Network (DBN). However, before we can define a DBN, we need to define the Restricted Boltzmann Machine (RBM).

2.2.1 Restricted Boltzmann Machine

The Restricted Boltzmann Machine, as descbribed in [Smo86], [Hino2] and [Hin12], is a two-layer network with symmetrically weighted connections. The two layers are often referred to as the "visible" and "hidden" layer. The "visible" layer serves as the input layer and the "hidden" layer is to be trained as a feature detector.

For training the RBM we use Contrastive Divergence (CD) which was proposed in [Hino2]. For CD_n we follow the following basic scheme:

1. Set the visible units to a randomly selected training vector ($\mathbf{v}$).
2. Compute, in parallel, for each hidden unit (h_j) its total input and use this to calculate the probability that the unit will be activated. For this calculation we use

$$p(h_j = 1|\mathbf{v}) = \sigma(b_j + \sum_i v_i w_{ij}),$$

   where b_j is the bias and $\sigma(x)$ is the logistic sigmoid function $1/(1 + exp(-x))$. Given the calculated probabilities, sample the binary state of each hidden unit.
3. "Reconstruct" the visible units, based on the computed hidden units in the same manner but with the roles reversed.
4. Update the weights based on the "reconstruction" and the learning rate (ϵ) using

$$\Delta w_{ij} = \epsilon(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{recon}),$$

   where $\langle v_i h_j \rangle_{data}$ represents the states of v_i and h_j as computed in step 2 and $\langle v_i h_j \rangle_{recon}$ is the state after reconstruction (step 3).
5. Repeat steps 2-4 n times.

Performing this scheme for every instance of a dataset, should allow the hidden layer to become an effective feature detector. A single run through the scheme, CD_1, is also referred to as a step of alternating Gibbs sampling. So we can say that CD_n performs n Gibbs steps.

2.2.2 Deep Belief Network

Deep Belief Networks are deep feedforward networks that are built by stacking RBMs in a greedy manner. This concept was first introduced in [HSo6]. We can build a DBN by first assuming the input layer is the visible layer of a RBM and the first hidden layer is its hidden layer. After training this first RBM, we can view the first hidden layer as the visible layer of the next RBM and the second hidden layer as its hidden layer. We can continue in this manner until we have built our desired network. Figure 2.2 gives an illustration of the layer-wise training of the DBN. We will be referring to this process as pre-training.

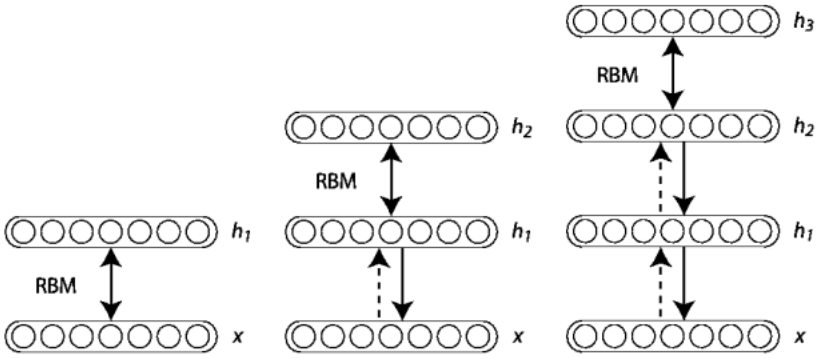

Figure 2.2: Layer-wise training of a DBN [Ben]

During pre-training, training instances are generated by putting the instances of the dataset through the network. Because the network ends at the visible layer of the RBM that you are training at that time, you can use the output of the network as instances for training the RBM. Pre-training is a unsupervised training method, because no knowledge of the class of a training instance is required.

After pre-training a DBN, it is possible to perform fine-tuning. This is often done with supervised training methods like backpropagation with gradient descent [Ama93]. The implementation of the DBN by deeplearning.net, which we used for our experiments, uses stochastic supervised gradient descent with minibatches for fine-tuning. We call such a method supervised because it uses the knowledge of what class an instance belongs to. Backpropagation uses this knowledge to determine how far off the prediction is from the correct result, i.e. the error. It then propagates this error back through the network during which it updates the weights between the layers to better predict that instance. Stochastic gradient descent (SGD) refers to the method by which the weights are updated. SGD attempt to repeatedly take small steps down on an error surface defined by a loss function. The gradient, which determines the movement on the error surface, is estimated using the error of just a few or a single instance. Using minibatches means that, instead of updating the weights based on the gradient estimated for each instance, we update based on the avarage gradient of a minibatch, a group of examples. The size of these minibatches are constant throughout training.

Algorithm 2 outlines the entire training process, where D is the dataset, N is the desired network, PE are the pre-training epochs, FE are the fine-tuning epochs, PL is the pre-training learning rate, FL is the fine-tuning learning rate and k is the number of Gibbs steps to do in CD_k.

Algorithm 2 DBN(D, N, PE, FE, PL, FL, k).

1: DBN = []; (contains the set of trained weights)
2: **for** $l = 0$ **to** $L_N - 2$ **do**
3: initialize W_l;
4: RBM = construct_RBM(N, l, W_l); (constructs a RBM with layer l of network N as visible layer using W_l)
5: **for** $epoch = 1$ **to** PE **do**
6: pre-train(RBM, D, PL, k); (trains W_l, with learning rate PL, on dataset D, using CD_k)
7: DBN.append(W_l);
8: **for** $epoch = 1$ **to** FE **do**
9: fine-tune(DBN, D, FL);

We chose to use a DBN, because [BLPL06] showed that deep networks, without pre-training, perform worse than shallow networks. They also showed that unsupervised greedy layer-wise pre-training, like that of a DBN, can perform significantly better than purely supervised greedy layer-wise pre-training. Therefore we believe that the DBN will be a good classifier.

Chapter 3

Related Work

In this chapter we discuss related work. We will discuss research that has been conducted in the field of frequent graph mining and molecular activity prediction. We will also look at other uses of Deep Belief Networks.

3.1 Frequent graph mining and molecular activity prediction

In [WMFP05] the frequent graph mining methods MoFa [BB02], FFSM [HWP03], gSpan [YH02] and Gaston [NK04] were compared. Each of these four algorithms only finds connected subgraphs and searches in a depth-first order. These methods were compared by implementing them in a common code base with the same level of programming expertise and optimization. The experiments showed a more or less clear runtime ranking where Gaston was generally the fasted algorithm followed by gSpan, FFSM and last MoFa. They also showed that gSpan requires the least memory. [FM04] provided an overview of some graph mining methods and extensions to improve on them when it comes to molecular data mining.

Although drug discovery is one of the main applications of graph mining, little focus has been placed on how to apply graph mining methods in the accurate classification of molecules. There are of course a few exceptions such as [KNK+06]. In this article a chemical representation was developed that allowed the graph mining algorithm Gaston to consider general and/or highly specific features. The authors then applied data mining to the result set to extract six nonredundant, discriminative substructures that represented relevant biochemical knowledge. Using a decision list comprised of these substructures they were then able to classify with decent accuracy. In [HCKR04] the two step procedure of graph mining and data mining was used to show that molecular substructures allow more accurate prediction of activity than molecular properties. The molecular feature miner Molfea, introduced in [HKDR03] and specifically designed for the mining of molecules, was used to generate the substructures. The data mining was performed with C4.5, J48, PART and SVM. In most cases the support vector machine performed best. gBoost [SNK+09] is a method that combines the two processes into one. It uses methematical programming with LP-Boost [DBS02] as a base algorithm combined with gSpan to learn from graph data. At each iteration a new substructure is added to the solution which should improve its classification accuracy.

Molecular activity prediction is not only being investigated through graph mining nor is it only used for drug discovery. For example: [WLAA95] describes the structure and scientific bases of an expert system meant to help predict the possible toxicity of new chemicals; [KMSS96] used inductive logic programming and a chemical structure represented by atoms and their bond connectivities; [VPH+04] applied three QSAR

methods for prediction of Ames genotoxicity; [LUY+05] tested several statistical learning methods using molecular descriptors (topological, quantum chemical and geometrical) as features of a molecule. This shows how big of a research field molecular activity prediction really is.

3.2 Deep Belief Networks

Applying deep belief networks has been the subject of a number of recent studies, one of which is [SHD14]. In this studies the application of DBNs to a natural language understanding problem was compared to the application of support vector machines (SVM), boosting and maximum entropy. The study showed that DBNs produce better classification results than maximum entropy and boosting based classifiers. The results of SVMs were however nearly identical.

Another study that applied DBNs is [MDH12], which applies them to acoustic modeling. In this study it is shown that DBNs outperform previously reported results on the TIMIT dataset, including those of Gaussian mixture models which were the dominant technique in acoustic modeling. [HDY+12] discussed the views of four research groups regarding the use of DBNs for acousting modeling in speech recognition. DBNs were also used for voice activity detection [ZW13] and speech synthesis [LDY13].

In [NH09] DBNs were used for 3D object recognition. Performance was evaluated on the NORB database and DBNs were shown to outperform models such as SVMs.

In 2012 the Merck Molecular Acitivy Challenge [Inc] was held which challenged participants to predict molecular activity. The participants were given 15 molecular data sets and descriptors derived from the molecule's chemical structure. The challenge was won by a team that employed a deep learning strategy. The winning team described in [Dah] that they used three essential components: single-task neural networks, multi-task neural networks and Gaussian process regression. The fact that a deep learning based solution won this challenge suggests that deep learning is likely to be an effective tool for predicting molecular activity. Unlike the participants of this challenge we first use graph mining to automatically generate subgraphs (descriptors) before using a DBN to classify instead of having experts determine the descriptors.

Chapter 4

Contributions

In this chapter we will discuss what changes we made to the implementation of gSpan. We will also describe how we combined gSpan and the DBN to be able to classify a molecular data set.

4.1 gSpan

The implementation of gSpan, provided by S. Nijssen, which was used in [BZRN06], finds the subgraphs that best divide the data set. The results of this search however often lead to many subgraphs that are very similar. Because similar subgraphs tend to classify the same instances, the results of this gSpan implementation will likely only cover a limited set of instances of the data set. Attempting to classify with such a set of subgraphs would likely lead to poor results. It is for that reason that we decided to implement two additional search methods, sequential coverage and a tree search. These two methods are meant to find the best subgraphs for classification.

4.1.1 Sequential Coverage

The goal of sequential coverage is to find a set of solutions, in our case subgraphs, which covers every instance of a dataset. It does so by running the search algorithm, for us gSpan, multiple times. After each run it deletes all instances of the dataset that are covered by the subgraphs that were found. This process continues until there are no instances, or at most *minSize* instances, left. The sequential coverage process is visualized in Figure 4.1.

Applying sequential coverage in this form to the same dataset would always result in a result set of the same size. For our purposes however, we would like to be able to specify ourselves what the size of the resultset should be. Therefore, if the result set that was found is smaller than desired, we extend the standard sequential coverage by repeating the process until the desired result set size is reached. We will refer to this as *multiple sequential coverage*. When we reach the desired size of the result set we stop the sequential coverage process.

When applying sequential coverage, we call gSpan several times. Normally each run would give us the same set of results, which is undesirable. Therefore we have extended gSpan to ignore the graphs of a provided set as potential results, but do not exclude their children. If we call this set of previously found graphs R, we could replace line 3 in subprocedure 1 by

3: **if** $s \notin R$ **then**
4: $\quad S \leftarrow S \cup \{s\}$;

Figure 4.1: Sequential Coverage visualized with Venn diagrams

Algorithm 3 outlines the pseudo-code for our implementation of multiple sequential coverage, where *GS* is the graph dataset, *S* is the resultset, *size* is the desired size of the resultset and *stepsize* determines how many subgraphs gSpan should find at a time.

Algorithm 3 Multiple_Sequential_Coverage(*GS*, *S*, *size*, *stepsize*)

```
1:  found ← 0;
2:  R^1 ← ∅; (results from previous sequential coverages)
3:  while found < size do
4:      T ← GS;
5:      R ← ∅; (results of the current sequential coverage)
6:      while |T| ≥ minSize and found < size do
7:          S ← ∅;
8:          gSpan(T,S,stepsize,R ∪ R^1);
9:          if |S| = 0 then
10:             break;
11:         else
12:             R ← R ∪ S;
13:             found = found + |S|;
14:         T ← GS − {g|g ∈ s.GS, s ∈ R};
15:     R^1 ← R^1 ∪ R;
16: S ← R^1;
```

The algorithm performs multiple sequential coverage by using R as the set of results of the current sequential coverage and remembering the results of previous coverages in R^1. We end a sequential coverage and move its results from R to R^1 (line 15) when the coverage is complete (line 6), the desired result set size is found (line 6) or if no new subgraphs could be found for the remaining instances (line 9,10). On line 8, when we call gSpan, we combine the sets R and R^1 to be the set of all subgraphs found so far, so that gSpan can refrain from finding those subgraphs again. At the end (line 16) all results found over all sequential coverages are added to the result set S.

4.1.2 Tree search

Although multiple sequential coverage provides good coverage of the data set, it discovers subgraphs that cover both positive and negative instances without providing a way to classify amongst those instances. We therefore introduce the tree search algorithm which attempts to do exactly that. [FZC+08] formulated an algorithm, similar to that in [BZ05], for building a model-based search tree. This algorithm also returned the set of features, which for our use would be frequent subgraphs, used in the tree. It is this set of features that we wish to obtain. The main idea of this algorithm is to recursively mine a discriminative feature that divides a subset of instances into purer subspaces that previously chosen patterns fail to distinguish. This process continues until either a subset is too small or consists only of instances of the same class. Figure 4.2

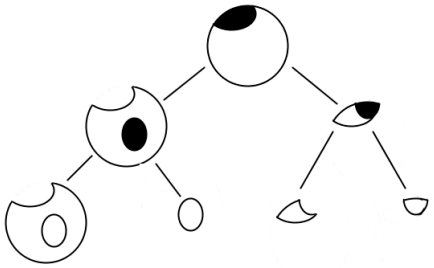

Figure 4.2: Tree search visualized with Venn diagrams

shows a part of the resulting tree from the viewpoint of the dataset. [FZC+08] showed that this algorithm could mine predictive patterns with extremely low global support. This means that, using this algorithm, we can discover subgraphs that do well in classification but that are not frequent when we consider the entire dataset. These subgraphs would therefore not be found with the previously discussed search methods.

Because we want to be able to specify the size of the result set and still get the best predictive subgraphs, we will abandon the recursive nature of the algorithm. Instead we will use a queue of paths that can be expanded further. The position of a path in the queue is determined by the amount of instances in its subset. The first path in the queue is the path with the most instances. We chose this order, because we expect a larger dataset to be more likely to lead to a more discriminative feature. As a result we can formulate algorithm 4, where GS is the graph dataset, S is the result set and *size* is the desired size of the result set.

Algorithm 4 Tree_Search(GS,S,*size*)

1: $found \leftarrow 0$;
2: $R \leftarrow \varnothing$; (contains elements of the tree)
3: $Q = \{\varnothing\}$; (queue of paths to expand)
4: **while** $found < size$ **and** $|Q| \neq 0$ **do**
5: $path \leftarrow$ path in Q with the most instances in its leaf;
6: $T \leftarrow GS - \{g | g$ not covered by any pattern in the $path\}$;
7: gSpan(T,S,1,R);
8: **if** $|S| \neq 0$ **then**
9: $S.GS \leftarrow \{g | g$ covered by a pattern in $S, g \in T\}$;
10: **if** $support(S) \geq minSize$ **and** $S.GS$ does not contain only positive or negative instances **then**
11: $Q \leftarrow Q \cup (path \cup S, +)$;
12: **if** $|T| - support(S) \geq minSize$ **and** $T - S.GS$ does not contain only positive or negative instances **then**
13: $Q \leftarrow Q \cup (path \cup S, -)$;
14: $R \leftarrow R \cup S$;
15: $found = found + 1$;
16: $Q \leftarrow Q - path$;
17: $S \leftarrow R$;

On line 3 the queue (Q) is initialized with the empty path, the root of the tree. New paths are added to the queue in lines 10-13, if their subsets of instances are larger than *minSize* and do not contain instances of a single class. When we are including the path $(path \cup S, +)$, the resulting subset of instances for this path is the set instances that are covered by the new subgraph (from S) and were in the subset of the original *path*. Thus $(path \cup S, -)$ extends the path by leaving those instances that are not covered by the new subgraph. Line 16 removes the path we have just expanded from the queue, even if no frequent subgraph was found ($|S| = 0$). Line 4 ensures that we either find the entire tree (empty queue) or find exactly *size* amount of subgraphs. At the end, line 16, we add all the found subgraphs to the result set.

4.2 Combining gSpan and the Deep Belief Network

In order to make predictions about a molecular data set using the DBN classifier, we first need a way of 'translating' this data set to a numerical dataset. We can do this by considering each molecule in the data set as a set of subgraphs that it does (1) or does not (0) contain. However if we were to consider every possible subgraph of every instance, we would get an enormous data set. Therefore we only use the subgraphs found by gSpan. This approach is also known as 'bag of features', where each subgraph is a potential feature of the molecule.

After running gSpan and obtaining a set of frequent subgraphs the process of creating the new data set is as follows:

1. We add a column for each of the graphs in the result set and for each instance of the molecular data set we add a row.
2. Each cell is then filled with a 1 if the graph is a subgraph of the molecule and is otherwise filled with a 0. When completed the set of 1's in each row represent the set of subgraphs that describe the molecule.
3. Finally an additional column is added to specify what class the instance belongs to.

The newly formed data set can then be used as input for the DBN. We will use this method to create data sets for each of the introduced search methods. This will allow us to compare their predictive capability.

Chapter 5

Evaluation

In this chapter we will discuss our experiments and their results. In our experiments we will compare the introduced search methods, optimize the DBN variables and compare the performance of DBNs with other classifiers.

5.1 Experiments

The molecular data set used in our experiments was retrieved from the authors of [KNK+06]. It is a mutagenicity data set with 4337 entries. Each compound in this data set was classified as a nonmutagen if only negative Ames test results were reported for it and as mutagen if at least one positive Ames test result was reported.

In our experiments we have only used uniform networks, networks where every hidden layer is the same size. This type of network however might not lead to the best results. Because the same simplification was used in [MDH12] and [NH09] we hope uniform networks suffice to show whether the greedy layerwise pretraining of a deep belief network improves classification results. In order to guarantee fair comparison of the results we have set the finetuning epochs at a constant 500 with a learning rate of 0.1 and a batch size of 10. During finetuning the best result, of the validation set, is stored and serves as the final prediction.

All experiments were run using 10-fold cross-validation. Experiments on the same data set ran on the exact same folds to ensure a fair comparison. Experiments with other classifiers were also run on the same folds. A '-' in a results table indicates that the experiment had a runtime longer than 2 days and was unlikely to provide additional insight.

5.1.1 Graph mining search method

In this thesis we have introduced three search methods that can construct a numerical data set from a molecular data set using gSpan. These three methods are the original implementation as provided, multiple sequential coverage and tree search. We want to know what search method results in the best classifcation when classifying with DBNs. To test this we first constructed six data sets. Each search method generated two of these data sets with one data set containing 100 features and the other 310 features. In case of the tree search the larger dataset represents a full tree. For each of the data sets we ran tests with 1-to-5 layers, layer sizes of 200 and 500 and for 0, 100 and 1000 pretraining epochs. The low amount of layers are meant to facilitate fast results while still giving a good indication of potential. We tested for 200 and 500 layer sizes because in our preliminary tests these seemed to do well. The pre-training learning rate was kept constant at 0.01 and only

Dataset: Normal 100

N	PE	L = 1	L = 2	L = 3	L = 4	L = 5
	0	28.511628	28.465116	28.674419	28.651163	28.790698
200	100	28.767442	28.744186	28.860465	28.837209	28.790698
	1000	28.651163	28.976744	29.139535	29.000000	28.930233
	0	28.674419	28.720930	28.744186	28.651163	28.837209
500	100	28.697674	28.837209	28.604651	28.697674	28.651163
	1000	28.558140	28.953488	28.720930	28.790698	-

Dataset: Normal 310

N	PE	L = 1	L = 2	L = 3	L = 4	L = 5
	0	27.744186	27.465116	27.488372	27.418605	27.418605
200	100	27.651163	27.674419	27.558140	27.651163	27.581395
	1000	27.488372	29.093023	29.325581	29.069767	29.116279
	0	27.441860	27.581395	27.534884	27.465116	27.325581
500	100	27.581395	27.627907	27.488372	27.325581	27.465116
	1000	27.697674	28.302326	27.651163	28.069767	-

Dataset: Multiple Sequential Coverage, 100 features

N	PE	L = 1	L = 2	L = 3	L = 4	L = 5
	0	17.046512	16.813953	17.116279	16.651163	16.325581
200	100	17.279070	17.744186	17.488372	17.720930	17.302326
	1000	17.790698	17.674419	17.720930	17.790698	17.302326
	0	17.046512	16.674419	17.186047	17.186047	17.000000
500	100	17.209302	18.348837	17.488372	17.720930	17.162791
	1000	17.325581	17.279070	17.534884	17.302326	-

Dataset: Multiple Sequential Coverage, 310 features

N	PE	L = 1	L = 2	L = 3	L = 4	L = 5
	0	15.930233	15.790698	15.534884	15.139535	15.186047
200	100	16.465116	16.186047	16.162791	16.372093	16.534884
	1000	16.837209	15.837209	17.302326	17.116279	17.511628
	0	15.953488	15.860465	15.465116	15.534884	15.186047
500	100	16.372093	16.348837	16.511628	16.255814	16.279070
	1000	16.139535	16.697674	16.720930	16.581395	-

Dataset: Tree search, 100 features

N	PE	L = 1	L = 2	L = 3	L = 4	L = 5
	0	16.930233	16.790698	16.348837	16.116279	15.883721
200	100	17.000000	17.162791	16.953488	16.976740	16.488372
	1000	17.372093	17.558140	17.232558	17.279070	16.976744
	0	16.697674	16.813953	16.534884	16.581395	16.325581
500	100	16.860465	17.116279	17.279070	17.139535	16.744186
	1000	17.023256	17.930233	17.209302	17.465116	-

Dataset: Tree search, 310 features

N	PE	L = 1	L = 2	L = 3	L = 4	L = 5
	0	14.651163	14.255814	14.116279	13.720930	13.813953
200	100	15.162791	15.000000	15.023256	14.790698	14.837209
	1000	15.418605	15.000000	16.046512	16.441860	16.465116
	0	14.511628	14.232558	14.302326	13.883721	13.813953
500	100	15.046512	15.069767	15.441860	15.534884	14.837209
	1000	15.162791	15.232558	15.837209	16.116279	16.046512

Table 5.1: Error rates for L layers of size N with PE pre-training epochs

one Gibbs step (CD_1) was used in pre-training. The resulting error rates of these experiments are shown in Table 5.1.

In Table 5.1 we can see that, the data set with 310 features generated by tree search, results in the lowest error rates. We can also see that data sets with 310 features generally outperform those with 100 features. Furthermore tree search seems to consistently peform better than multiple sequential coverage when only comparing data sets with the same amount of features. The 'normal' search method results in the highest error rates and thus performs the worst. Based on these results the remainder of the experiments will only be run on the data set of 310 features generated by tree search. Neither the 200 layer size nor the 500 layer size seemed to outperform the other. We will therefore continue the remainder of our experiments with layer sizes of 200, because these should be faster to train.

5.1.2 Optimizing the deep belief network variables

Having selected a dataset we will now try to find optimal values for some of the variables of the deep belief network. The three variables we are interested in are the amount of layers, the amount of Gibbs steps during pre-training and the pre-training learning rate. For all other variables, except pre-training epochs, we have already stated their constant value.

We start by trying to find the amount of layers that lead to the best classification results. Although it is possible that the optimum lies beyond 15 layers, we have only tested up to 15 layers. During these tests the pre-training learning rate was set at 0.01 and CD_1 (1 Gibbs step) was used for pre-training. All tests were run with 0, 100 and 1000 pre-training epochs. The error rates resulting from these experiments are shown in Table 5.2.

L	PE = 0	PE = 100	PE = 1000
1	14.651163	15.162791	15.418605
2	14.255814	15.000000	15.000000
3	14.116279	15.023256	16.046512
4	13.720930	14.790698	16.441860
5	13.813953	14.837209	16.465116
6	13.767442	15.186047	16.906977
7	13.953488	14.697674	17.069767
8	14.093023	14.604651	17.186047
9	13.697674	14.325581	16.697674
10	13.860465	14.279070	14.604651
11	13.813953	14.162791	14.627907
12	14.069767	14.465116	14.813953
13	13.953488	13.976744	16.627907
14	13.930233	20.906977	23.395349
15	46.860465	20.674419	24.093023

Table 5.2: Error rates for L layers of 200 size with PE pre-training epochs

From Table 5.2 we can see that the lowest error rate is with nine layers and 0 pre-training epochs. Because a DBN without pre-training simply equals a MLP we are also interested in what layers perform best with pre-training. We can see that with 1000 pre-training epochs ten layers lead to the best results. Therefore during the remainder of our experiments we will only consider nine and ten layers.

Next we wanted to know if increasing the number of Gibbs sampling steps during pre-training improves classification. To test this we looked at 1, 5, 10 and 15 Gibbs steps for nine and ten layers. These experiments were run with 0, 100 and 1000 pre-training epochs. The error rates resulting from these experiments are shown in Table 5.3. The results suggest that using a single Gibbs sampling step results in the lowest error

rates. In Table 5.4 the execution times are given for a single fold of nine layers with 1000 pre-training epochs. From this table we can also see that increasing the number of Gibbs sampling steps greatly impacts execution time. The lack of improvement and the impact on execution time that comes from increasing the amount of Gibbs steps therefore leads us to choose a single Gibbs step as the optimum.

L	PE	k = 1	k = 5	k = 10	k = 15
9	0	13.697674	13.697674	13.697674	13.697674
	100	14.325581	14.883721	14.860465	14.837209
	1000	16.697674	19.139535	19.023256	-
10	0	13.860465	13.860465	13.860465	13.860465
	100	14.279070	14.813953	14.604651	14.534884
	1000	14.604651	17.953488	18.093023	18.860465

Table 5.3: Error rates for L layers of 200 size with PE pre-training epochs with k Gibbs steps

L	PE	k = 1	k = 5	k = 10	k = 15
9	1000	9158	37956	67807	96042

Table 5.4: Execution times in seconds for pre-training a single fold with k Gibbs steps for L layers with PE pre-training epochs

Finally we want to know if increasing the pre-training learning rate has a positive effect on classification results. To test this we ran experiments for pre-training learning rates of 0.01 and 0.05. The experiments were run with 9 layers of size 200 and with one Gibbs step. The error rates resulting from these experiments are shown in Table 5.5.

PE	PL = 0.01	PL = 0.05
0	13.697674	13.697674
100	14.325581	16.860465
1000	16.697674	20.023256

Table 5.5: Error rates for PE pre-training epochs with PL pre-training learning rate

Table 5.5 shows that a learning rate of 0.01 performs best. Therefore we choose a pre-training learning rate of 0.01 as our optimized value.

5.1.3 Comparing performance of DBNs with other classifiers

The experiments so far have served to find the best classification results for a uniform DBN. We are however most interested in how DBNs perform compared to other classifiers. And as such we will now compare our optimized DBN to some WEKA classifiers. The WEKA classifiers chosen for this are Naive Bayes, Random Forest, Rotation Forest, Adaboost M1 and SMO. We optimized each of the chosen classifiers for this data set. We also compare DBNs against a multi-layer perceptron (MLP) by using the result obtained with 0 pre-training epochs. After all a MLP is a DBN without any pre-training.

We optimized the WEKA classifiers for the data set by using the WEKA Experimenter. Optimization, unlike for DBN, was not done on the same 10 folds as the experiments, instead we used the 10-fold cross-validation functionality build into WEKA. We optimized one variable at a time by running the WEKA Experimenter for a range of values of that variable while keeping the remaining variables constant. For example, while optimizing the numTrees variable of Random Forest, we ran 10 iterations of 10-fold cross-validation for numTrees values of 10, 20, 30, 40 and 50 while keeping maxDepth at 0, numFeatures at 0 and seed at 1. After optimizing a variable its optimized value was used for the remaining optimizations for that classifier.

Classifier	Variable	Value
Random Forest	maxDepth	0
	numFeatures	0
	numTrees	40
Rotation Forest	classifier	Random Forest with numTrees 40
	maxGroup	9
	minGroup	6
	removedPercentage	50
AdaBoost M1	classifier	J48 with 0.15 confidence factor
	weightThreshold	100
SMO	kernel	NormalizedPolyKernel with exponent 3.0
	c	1.0

Table 5.6: Optimized variables of WEKA classifiers and their value

A variable was only changed when there was a statically significant improvement and not all variables were extensively tested. The variables that were optimized and their optimized values are given in Table 5.6

We ran experiments over the same 10 folds for each of the optimized classifiers and DBNs with 0 (MLP), 100 and 1000 pre-training epochs. The error rates and the standard deviation of the folds resulting from these experiments are shown in Table 5.7.

Classifier	Error rate	Std. Dev.
Naive Bayes	27.25174	1.4647
Random Forest	15.21941	1.1493
Rotation Forest	14.87299	1.5971
AdaboostM1	15.6813	2.2588
SMO	15.31179	1.7657
MLP	13.697674	1.7794
DBN (100 PE)	14.279070	1.9121
DBN (1000 PE)	14.604651	1.9296

Table 5.7: Error rates and standard deviation for WEKA classifiers and DBNs

Statistical analysis was performed on the results using a paired t-test with a 95% confidence level and using each fold as a datapoint. The P values resulting from these tests can be seen in Table 5.8. In this table values with a * indicate the classifier that makes up the row is better and $^\wedge$ indicates the column classifier is better. Statistical significance occurs at P values under 0.05.

	1	2	3	4	5	6
1. Random Forest						
2. Rotation Forest	0.341					
3. Adaboost M1	0.366	0.086				
4. SMO	0.825	0.248	0.223			
5. MLP	0.003^*	0.004^*	0.000^*	0.000^*		
6. DBN (100 PE)	0.030^*	0.111	0.000^*	0.006^*	0.039	
7. DBN (1000 PE)	0.256	0.468	0.023^*	0.091	$0.029^\wedge$	0.408

Table 5.8: P values for paired t-tests

The results shown in tables 5.7 and 5.8 indicate that, when using a DBN, the individual steps between 0 (MLP), 100 and 1000 PE are not statistically significant. However the difference between 0 PE and 1000 PE is statistically significant and therefore suggest that pre-training a uniform deep belief network does not improve classification of molecular data sets. Because the MLP also outperformed the WEKA classifiers we tested, we conclude that the MLP performs best of the tested classifiers in classifying molecular data sets.

5.1.4 Other molecular data sets

Up until now our experiments have focused on a single molecular data set. Conclusions drawn from just one data set are however not always representitive for all data sets. Therfore we also experimented with two other data sets. The first is also a mutagenicity data set and has 684 entries. The second is a HIV data set with 1486 entries. The graph mining step was performed with a tree search which continued until a full tree was found. For the mutagenicity data set this resulted in 64 features and for the HIV data set 90 features.

For the experiments the same optimized values of the WEKA classifiers were used as before. All DBN variables, except for the amount of layers, were also kept the same. Next to the optimal value of 9 layers in earlier experiments, we also experimented with 6 layers because a smaller data set might do better with a smaller network. All experiments, for one data set, ran on the same 10 folds for fair comparison and again the paired t-test with a 95% confidence level was used for statistical analysis. The resulting error rates, standard deviations and P values can be found in Tables 5.9 and 5.10.

Classifier	Error rate	Std. Dev.	2	3	4	5	6	7
1. Naive Bayes	27.64706	4.6400						
2. Random Forest	20.14706	5.4166						
3. Rotation Forest	19.26468	4.2424	0.424					
4. Adaboost M1	18.23529	3.2663	0.077	0.310				
5. SMO	19.26469	5.6084	0.343	0.999	0.382			
6. MLP (9 layers)	13.166667	3.3747	0.008*	0.002*	0.005*	0.009*		
7. DBN (100 PE, 6 layers)	18.333333	4.3744	0.116	0.083	0.925	0.257	0.008^	
8. DBN (1000 PE, 6 layers)	18.166667	5.1190	0.295	0.441	0.966	0.490	0.035^	0.879

Table 5.9: Error rates, standard deviation and P values for WEKA classifiers and DBNs for the mutagenicity data set

The results of the small mutagenicity data set shown in Table 5.9 indicate that the MLP provides the most accurate prediction. The differences between the other classifiers have no statistical significance. This means that for this smaller mutagenicity data set our previous conclusions hold.

Classifier	Error rate	Std. Dev.	2	3	4	5	6	7
1. Naive Bayes	19.05405	3.2917						
2. Random Forest	17.02703	1.5862						
3. Rotation Forest	17.56757	1.4244	0.168					
4. Adaboost M1	15.87836	1.6319	0.060	0.002*				
5. SMO	15.47297	1.0296	0.005*	0.001*	0.425			
6. MLP (6 layers)	12.857143	2.1560	0.000*	0.000*	0.001*	0.001*		
7. DBN (100 PE, 6 layers)	17.071429	4.5606	0.978	0.748	0.422	0.227	0.017^	
8. DBN (1000 PE, 6 layers)	14.142857	3.4634	0.007*	0.006*	0.105	0.140	0.137	0.034*

Table 5.10: Error rates, standard deviation and P values for WEKA classifiers and DBNs for the HIV data set

The results for the HIV data set given in Table 5.10 match, for the most part, the conclusions we have drawn for the previous data sets. However the error rate for 1000 pre-training epochs is not significantly worse than that of the MLP and it is significantly better than the error rate for 100 pre-training epochs. This could indicate that with increased pre-training effort a DBN might perform even better on this data set. The results from the experiments as discussed in this thesis however suggest it is unlikely. Therefore we hold to our conclusion that pre-training a uniform DBN does not improve classification results for molecular data sets.

Chapter 6

Conclusions

In this thesis we combined the graph mining method gSpan with Deep Belief Networks in an attempt to accurately classify a molecular data set. gSpan was used to find frequent subgraphs that could be seen as features of the molecules. Using the features found with gSpan we transformed the molecular data set into a numerical data set that could be used for classification with a DBN. With the aim of improving classification, we extended gSpan with two search methods, (multiple) sequential coverage and tree search. Our experiments showed that the features found with tree search were best suited for classification with DBNs.

DBNs are deep feedforward networks that can be pre-trained in a greedy layer-wise manner by stacking restricted boltzmann machines (RBM). After pre-training, DBNs can be fine tuned to optimize classification. However the pre-training of uniform DBNs (each hidden layer the same size) seems to have a negative effect on the classification of molecular data sets. This became evident from experiments on three seperate data sets in which the multi-layer perceptron (MLP), which we simulated by using the DBN without pre-training, outperformed DBNs that did apply pre-training. The WEKA classifier Naive Bayes always performed worst, while WEKA classifiers Random Forest, Rotation Forest, Adaboost and SMO were consistently outperformed by the MLP.

Although our results suggest that DBNs are unlikely to yield the best classification results, we can not exclude the possibility that DBNs which are not uniform would perform better. The fact that the multi-layer perceptron outperformed all other classifier that we tested, does provide hope that the combination of graph mining and deep learning could lead to accurate molecular activity prediction. Research into the use of, for example, convolutional deep belief networks might therefore be of interest in the future.

Bibliography

[Ama93] Shun-ichi Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4):185–196, 1993.

[BB02] Christian Borgelt and Michael R. Berthold. Mining molecular fragments: Finding relevant substructures of molecules. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9-12 December 2002, Maebashi City, Japan* [DBL02], pages 51–58.

[Ben] Yoshua Bengio. Deep belief networks. `http://www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/DeepBeliefNetworks`. Accessed 26-06-2015.

[BLPL06] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In Bernhard Schölkopf, John C. Platt, and Thomas Hoffman, editors, *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*, pages 153–160. MIT Press, 2006.

[BZ05] Björn Bringmann and Albrecht Zimmermann. Tree^2 - decision trees for tree structured data. In Jorge et al. [JTB+05], pages 46–58.

[BZRN06] Björn Bringmann, Albrecht Zimmermann, Luc De Raedt, and Siegfried Nijssen. Don't be afraid of simpler patterns. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Knowledge Discovery in Databases: PKDD 2006, 10th European Conference on Principles and Practice of Knowledge Discovery in Databases, Berlin, Germany, September 18-22, 2006, Proceedings*, volume 4213 of *Lecture Notes in Computer Science*, pages 55–66. Springer, 2006.

[Dah] George Dahl. Deep learning how i did it: Merck 1st place interview. `http://blog.kaggle.com/2012/11/01/deep-learning-how-i-did-it-merck-1st-place-interview/`. Posted 1-11-2012, accessed 18-08-2015.

[DBL02] *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9-12 December 2002, Maebashi City, Japan*. IEEE Computer Society, 2002.

[DBS02] Ayhan Demiriz, Kristin P. Bennett, and John Shawe-Taylor. Linear programming boosting via column generation. *Machine Learning*, 46(1-3):225–254, 2002.

[FM04] Ingrid Fischer and Thorsten Meinl. Graph based molecular data mining - an overview. In *Proceedings of the IEEE International Conference on Systems, Man & Cybernetics: The Hague, Netherlands, 10-13 October 2004*, pages 4578–4582. IEEE, 2004.

[FZC+08] Wei Fan, Kun Zhang, Hong Cheng, Jing Gao, Xifeng Yan, Jiawei Han, Philip S. Yu, and Olivier Verscheure. Direct mining of discriminative and essential frequent patterns via model-based search tree. In Ying Li, Bing Liu, and Sunita Sarawagi, editors, *Proceedings of the 14th ACM*

SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008, pages 230–238. ACM, 2008.

[HCKR04] Christoph Helma, Tobias Cramer, Stefan Kramer, and Luc De Raedt. Data mining and machine learning techniques for the identification of mutagenicity inducing substructures and structure activity relationships of noncongeneric compounds. *Journal of Chemical Information and Modeling*, 44(4):1402–1411, 2004.

[HDY+12] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

[Hin02] Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002.

[Hin12] Geoffrey E. Hinton. A practical guide to training restricted boltzmann machines. In Grégoire Montavon, Genevieve B. Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade - Second Edition*, volume 7700 of *Lecture Notes in Computer Science*, pages 599–619. Springer, 2012.

[HKDR03] Christoph Helma, Stefan Kramer, and Luc De Raedt. The molecular feature miner molfea. In *Proceedings of the Beilstein Workshop 2002: Molecular Informatics: Confronting Complexity*, 2003.

[HS06] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[HWP03] Jun Huan, Wei Wang, and Jan Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003), 19-22 December 2003, Melbourne, Florida, USA*, pages 549–552. IEEE Computer Society, 2003.

[Inc] Kaggle Inc. Merck molecular activity challenge. `https://www.kaggle.com/c/MerckActivity`. Accessed 18-08-2015.

[JTB+05] Alípio Jorge, Luís Torgo, Pavel Brazdil, Rui Camacho, and João Gama, editors. *Knowledge Discovery in Databases: PKDD 2005, 9th European Conference on Principles and Practice of Knowledge Discovery in Databases, Porto, Portugal, October 3-7, 2005, Proceedings*, volume 3721 of *Lecture Notes in Computer Science*. Springer, 2005.

[KMSS96] Ross D King, Stephen H Muggleton, Ashwin Srinivasan, and MJ Sternberg. Structure-activity relationships derived by machine learning: The use of atoms and their bond connectivities to predict mutagenicity by inductive logic programming. *Proceedings of the National Academy of Sciences*, 93(1):438–442, 1996.

[KNK+06] Jeroen Kazius, Siegfried Nijssen, Joost N. Kok, Thomas Bäck, and Adriaan P. IJzerman. Substructure mining using elaborate chemical representation. *Journal of Chemical Information and Modeling*, 46(2):597–605, 2006.

[LDY13] Zhen-Hua Ling, Li Deng, and Dong Yu. Modeling spectral envelopes using restricted boltzmann machines and deep belief networks for statistical parametric speech synthesis. *Audio, Speech, and Language Processing, IEEE Transactions on*, 21(10):2129–2139, 2013.

[LGRN09] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 609–616. ACM, 2009.

[LUY+05] H Li, CY Ung, CW Yap, Y Xue, ZR Li, ZW Cao, and YZ Chen. Prediction of genotoxicity of chemical compounds by statistical learning methods. *Chemical research in toxicology*, 18(6):1071–1080, 2005.

[MDH12] Abdel-rahman Mohamed, George E Dahl, and Geoffrey Hinton. Acoustic modeling using deep belief networks. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1):14–22, 2012.

[NH09] Vinod Nair and Geoffrey E. Hinton. 3d object recognition with deep belief nets. In Y. Bengio, D. Schuurmans, J.D. Lafferty, C.K.I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1339–1347. Curran Associates, Inc., 2009.

[NK04] Siegfried Nijssen and Joost N. Kok. A quickstart in frequent structure mining can make a difference. In Won Kim, Ron Kohavi, Johannes Gehrke, and William DuMouchel, editors, *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004*, pages 647–652. ACM, 2004.

[NK05] Siegfried Nijssen and Joost N. Kok. Multi-class correlated pattern mining. In Francesco Bonchi and Jean-François Boulicaut, editors, *Knowledge Discovery in Inductive Databases, 4th International Workshop, KDID 2005, Porto, Portugal, October 3, 2005, Revised Selected and Invited Papers*, volume 3933 of *Lecture Notes in Computer Science*, pages 165–187. Springer, 2005.

[SHD14] Ruhi Sarikaya, Geoffrey E. Hinton, and Anoop Deoras. Application of deep belief networks for natural language understanding. *IEEE/ACM Transactions on Audio, Speech & Language Processing*, 22(4):778–784, 2014.

[Smo86] Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory. 1986.

[SNK+09] Hiroto Saigo, Sebastian Nowozin, Tadashi Kadowaki, Taku Kudo, and Koji Tsuda. gboost: a mathematical programming approach to graph classification and regression. *Machine Learning*, 75(1):69–89, 2009.

[VPH+04] Joseph R Votano, Marc Parham, Lowell H Hall, Lemont B Kier, Scott Oloff, Alexander Tropsha, Qian Xie, and Weida Tong. Three new consensus qsar models for the prediction of ames genotoxicity. *Mutagenesis*, 19(5):365–377, 2004.

[WLAA95] Yin-Tak Woo, David Y Lai, Mary F Argus, and Joseph C Arcos. Development of structure-activity relationship rules for predicting carcinogenic potential of chemicals. *Toxicology letters*, 79(1):219–228, 1995.

[WMFP05] Marc Wörlein, Thorsten Meinl, Ingrid Fischer, and Michael Philippsen. A quantitative comparison of the subgraph miners mofa, gspan, ffsm, and gaston. In Jorge et al. [JTB+05], pages 392–403.

[YH02] Xifeng Yan and Jiawei Han. gspan: Graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9-12 December 2002, Maebashi City, Japan* [DBL02], pages 721–724.

[ZW13] Xiao-Lei Zhang and Ji Wu. Deep belief networks based voice activity detection. *Audio, Speech, and Language Processing, IEEE Transactions on*, 21(4):697–710, 2013.