POLITECNICO DI MILANO

School of Industrial and Information Engineering  
Department of Electronics, Information and Bioengineering  
Master of Science Degree in Computer Science and Engineering



BAYNESIAN GRAPH CONVOLUTIONAL NEURAL NETWORK WITH UNCERTAINTY ESTIMATION TO PREDICT MUTAGENICITY OF CHEMICALS

Supervisor:

Prof.Ing.Matteo Matteucci

Co-Supervisor:

Prof.Giuseppina GINI

Thesis of :

Chiakang HUNG 894087

ACADEMIC YEAR 2020-2021

Chiakang H.

*BAYNESIAN GRAPH CONVOLUTIONAL NEURAL NETWORK WITH UNCERTAINTY ESTIMATION TO PREDICT MUTAGENICITY OF CHEMICALS*

*2020*

[Chiakang.hung@mail.polimi.it](mailto:Chiakang.hung@mail.polimi.it)

Politecnico di Milano

*Computer Science Department*

Session Laurea: 1st October 2020

*For my Family*

**Abstract**

Most current state of arts for toxicity lack uncertainty estimation but uncertainty estimation is important for results interpretation since the prediction may be inaccurate if the model does not have sufficient of such data. Our models provide two results: one for uncertainty estimation and one without uncertainty estimation.

Graph convolutional neural network is a new architecture that takes the input as a graph and thus does not to compute or select descriptors. The graph convolutional neural networks are used for largest available dataset: the Ames data set, and the dataset from Japanese National health institute and science(NIHS). Using the Japanese NIHS data allows to create a model more oriented to industrial chemicals. The Japanese competition provides a different training data and allows to use a model of choice to predict on their test dataset. Our model will enter the competition in the end of 2020.

Our models provide performance results for different chemical classes to show which chemical class are better predicted and which has lower uncertainties. This information can be used by the researchers. To provide an explanation of the results obtained by previous literature, the results are validated with expert knowledge using different performance parameters. The expert knowledge is the structural alerts selected and provided by experts from Toxtree.

In the previous literature, the structural alerts can not be visualized algorithmically, in this work the structural alerts can be automatically extracted and visualized in the graph with short computational time. The models are fast to train and the overall performance of the model are in line with the current state of arts.

**Chapter 1**

**Introduction**

This thesis in about computational toxicology, and in particular about modelling the mutagenicity property of chemical substances.

Computational toxicology is a large spectrum of methods aimed at assessing the toxicity/safety of chemical substances using computers and models instead of wet experiments. Computational toxicology is the result of two research lines: mathematical chemistry and the 3Rs (Replace, Reduce, Refine,) principle, defined in the fifthies of the last century, and devoted to find ways to reduce animal use in assessing chemicals.

Mathematical chemistry (Basak 2013) and chemoinformatics, started with the definition of molecular structures and mainly worked on defining numerical chemical descriptors, i.e. human engineered features representing various characteristics as geometry, topology, and energy distribution of molecules. In a few decades of research, thousands of molecular descriptors have been defined, usually in relation with the rapid development of predictive methods for toxicology and new molecules design (Todeschini et al. 2009).

Chemical properties can be studied using physical models and simulation. Those methods are common for the design of new molecules, but are not affordable when the properties of interest are biological effects; in fact, knowledge about the living systems (for instance the receptor and the mechanism used to cause the effect) is not available. For modeling biological effects, (Quantitative) Structure-Activity Relationship - (Q)SAR - models are commonly used. QSARs are based on the postulate that similar molecules exhibit similar physical and biological activity (Johnson et al. 1990), and are the main non-physical predictive models in use (Gini 2016).

QSAR methods built using molecular descriptors appeared around the middle of the last century. Those models were initially simple regressions, using very few, and possibly simple, chemical descriptors. Choosing informative descriptors for the task at hand is a key aspect and requires deep insights into chemical and biological properties, as interactions between molecules, reactions and enzymes involved, and metabolic degradation of the molecules should be somehow represented.

Usually building QSARs starts with computing a large number of chemical descriptors, then applying a method to reduce them to a few, and adopting a computational technique to build the model that predicts new assays.

SAR is instead based on the idea that specific functional groups, often defined by experts and called structural alerts (SAs), are responsible for the molecular behavior, so their presence alone is checked to predict the activity. In SAR molecules are similar if they share the same functional groups. SAR and QSAR are usually mixed, as many QSAR methods use fingerprints and often the explanation of QSAR models considers the functional subgroups too.

For regulatory purposes, toxicologists use simple methods like read across, which are accepted, and often integrated in final decisions by panels of experts. In read across the target molecule is compared with a couple (or a few) of similar molecules for which the test has been performed, and the property value is predicted.

The problem with this method is human bias. Different experts select different similar molecules, or look for different substructures as responsible of the toxicity, or give a different weight to pysicochemical properties. Some experts might agree on some features, while other experts agree on others. (Benfenati el al. 2016)

Human bias when using QSAR models that learn from a large population of molecules. However, two factors make QSARs not widely accepted: the resistance of experts to the acceptance of statistical methods, and the shortage of data available that makes it problematic to cover the entire chemical space.Toxicity data are available for just a very small fraction of the entire chemical space of the molecules of biological interest. It has been estimated that this chemical space exceed 1060 molecules (Kirkpatrick et al. 2004), while the toxicological characterization of chemicals is available for a few thousand molecules.

Laboratory tests are expensive, time consuming, and in some cases forbidden by regulations (for instance, in Europe cosmetic ingredients cannot be tested on animals; in USA the testing programs on animals are largely substituted by in vitro tests). This situation pushes for the adption of computational methods that make use of available data and knowledge, and that can integrate results from livig organisms and cellular lines. Adopting (Q)SAR becomes a necessity, and making QSARs effective is more and more pursued.

Quite recently Machine Learning algorithms are being adopted to replace the simple statistical methods. In the last decade very big Neural Networks, called deep neural networks (DNN) gained attention in computational toxicology. DNNs using molecular descriptors were the top models in Merck Kaggle challenge in 2012 and in Tox21 challenge in 2014 for activity prediction.

This thesis explores the use of DNNs to solve some of the open issues in computational toxicology.

**1.1 The Challenges**

There are at least four major challenges to obtain toxicology models that can replace animal experiments.

The first problem is that usual (Q)SARs produce a prediction but lack extimating the uncertainty of the prediction; to make risk assessment uncertainty estimation plays an important role. Without uncertainty estimation, the model can only predict a number (for instance a dose) or a class label (for instance mutagenic or non-mutagenic) based on the knowledge used by the model. However, the classifier cannot tell you if it has sufficient knowledge to classify such data. If the data sample is out of the domain of the classifier, wrong prediction can occur, but if the classifier is able to identify how confident it is at the prediction, the users can choose whether to use this result or not, considering the uncertainty estimation and the cost of using wrong classifications. Even though most (Q)SAR do not offer uncertainty estimation, uncertainty is estimated in other applications, for instance in image classification. For example, Kendall and Gal(2017) provide confidence levels in classifying objects in the image. One target of the new model to develop is so to estimate how confident the classifier is in classifying each molecule.

A second challenge is to improve the process of constructing QSAR models. Instead of computing thosands of numerical descriptors and selecting a few ones as features used by the classifier, a simplified process would be to use directly as input a graph representation of the molecular structure, and let the algorithm to automatically extract the features. This mehod has been alreay tried considering the input as a string, i.e. the SMILES representation of molecules (Weininger et al ) or as a picture of the chemical graph. The idea here is to use instead the chemical graph, and this is possible using research on graph neural networks (REF).

A third challenge is to avoid, or to reduce, the bias of the expert that characterizes the SAR approach; usually the functional subgroups responsible for the activity are proposed by experts on the basis of the study of some mechanism of action. However, they are not a complete list, and are not universally applicable. The analysis of ToxTree (Benigni et al 2008), a public system based on criteria studied by the ISS (Istituto Superiore di Sanità – Italy) uses a few tens of functional subgroups as mutagenicity alerts, and indicates the percentage of their presence in toxic molecules; in some cases the presence is less that 50%. The role of SAs in fact is to give the expert a reason of toxicity in case this appears, but other mechanisms at organ or cellular level can make such alerts transformed into other chemicals before they act.

Here it comes is the last challenge for this thesis: to provide an explanation of the result obtained by the model (Gini 2018). Considering that toxicology models are used in many processes and regulations, some kind of explanation to ground the result obtaimed into a general expert knowledge is important. To this end existent tools can be applied to extract the features in the graph; the features can produce a first level of the explanation.

Other open problems, which affect this thesis but do not have a general solution, are data availability and data quality. To build a classifier it is necessary to collect data for the endpoint of interest. The common practice is to look up public databases to find experimental data. There are a few issues related to using a public database. A lot of databases are not open to the public, which limits the amount of data we can find. Also, different institutes have different rules and standards regarding the toxicity of a molecule; different regulations apply different protocols and the resulting values or labels can disagree in different databases. This may lead to extra efforts to create a reiable data set, as researchers have access and to understand the institute's methodology and standards. This problem has been skipped in this thesis since two dataset already collected have been selected; one obtained from previous research (Gini et al 2019), and a second one provided by the Japan Ministry of Industry as part of an international challenge (Honma et al 2019).

**1.2 Methodology**

Graph Convolutional Neural Network (Graph CNN) (Wu et al. 2019) and Bayesian Graph Convolutional Neural Network are extensions of deep neural networks, both rooted in the theory of graph neural networks.

Graph CNNs are generalizations of classical CNNs to handle graph data; for this reason they are of interest in handling molecular structures. Graph CNNs encode the nodes of the graph into an embedding space that approximates similarity in the original network. Mathematically they apply convolutions; while CNNs apply 2D convolutions to the image pixels, Graph CNN apply convolutions to node features.

Using graphs with GraphCNN for analyzing datasets has been tried in recent years. Using them for QSAR has not yet been tried. The reason is that a text representation for chemicals is very popular, for istance to create datasets, to make web search, and so on. This textual representation is called Simplified Molecular Input Line Entry Specification (SMILES). SMILES are ASCII strings obtained by printing the symbol nodes encountered in a depth-first tree visit of the chemical graph. SMILES are expressions of a context free language. The SMILES notation of a chemical compound is a string of atoms (represented by their atomic symbols), bonds, parentheses, and numbers. The four basic bond types are represented by the symbols ‘-‘, ‘=’, ‘#’, and ‘:’ (single and aromatic bonds may always be omitted), while ionic bonds are represented by a ‘.’. Branches are specified by enclosing brackets. Cyclic structures are represented by breaking one bond in each ring; the atoms adjacent to the bond obtain the same number. Hydrogen is not included in a SMILES representation, but can be inferred from the available valences. Typically, a number of equally valid SMILES can be written for a molecule. For example, CCO, OCC and C(O)C all specify the structure of ethanol. In conclusion SMILES contain exactly the same information of the chemical graph, but more SMILES strings can describe the same molecule (it depends on the way to represent the aromatic link, and to the atom used to start). Algorithms have been developed to ensure the same SMILES is generated for a molecule, and this is termed the canonical SMILES.

Graphs, on the other end, are uniquely representing a molecule, and this can be an advantage for the user, who will not care about canonical SMILES.

THIS PART TO BE EXTENDED WITH A SENTENCE ABOUT THE MAIN PAPERS USED.

Our models will explore different technologies in GCN, to improve the classification of toxic molecules and to provide explanations.

As before mentioned, uncertainty estimation is an important indicator of the quality of

classification of toxic molecules. Our model incorporates uncertainty estimation in classifying mutagenicity, inspired by the paper of (Ryu et al 2019), which applied it to the ChEMBL dataset.

**1.3 Chapters Overview**

In chapter 2 there is a presentation about experiments and available models for mutagenicity. The in vitro experiments, called Ames test, are described in detail. The experimental data available are also described, together with the main SAR and QSAR models so far developed to predict the output of the Ames test.

Chapter 3 deals with machine learning and deep learning methods, and in particular presents the deep networks used in our study.

Chapter 4 makes the design plan for the thesis; it discusses the motivation ad the main choices for designing new models for classifying mutagenicity.

Chapter 5 and 6 go in detail of how the new four models are built and tested. In particular, two models are built for each dataset, with and without uncertainty estimation.

Chapter 7 discusses the interpretability and other significances of the model. Chapter 8 is the conclusion of the work presented.

**Chapter 2**

**Mutagenicity**

Mutagenicity is the capacity of a substance to cause a permanent and transmissible change in the genetic material of cells or organisms. The changes may involve one or more genes or chromosomes. The mutagenicity property is of high public concern because it has a close relationship with carcinogenicity and potentially with reproductive toxicity.

For assessing the potential of a chemical to be mutagenic, different test procedures can be adopted, either in vivo or in vitro.

In vitro in Latin means in glasses. The term describes the studies of biological properties that are done in a dish or a test tube, containing cells from animals or human organs. In vitro contrasts with in vivo studies which are done using directly living beings. In vitro enables scientists to isolate specific cells, bacteria, and viruses for a closer look. This means the scientists do not have to be distracted by other factors that might affect the interpretation of the results. However, since the examination mainly focuses on test tube experiments instead on the actual animal, oftentimes the experimental results do not transfer well to real life. However, since the experiments do not sacrifice animals, this means in vitro is more ethical, less expensive, and safer. In vitro studies are common in drug design, where cells of the target organ are directly tested with the drug candidate. For assessing the environmental risk of chemicals in vitro studies are in their infancy. The most relevant study is Tox21, a collaborative project launched by EPA and NHI in USA that so far has tested about 10000 chemicals in 70 high-throughput assays. The obtained data set is under evaluation and has been used by an international challenge (https://tripod.nih.gov/tox21/challenge/)

In vivo tests focus on the response produced by the whole body. When the target is to assess human toxicity, mammals species, as mice and rats, are used to test adverse effects as well as toxicokinetics properties.

**2.1 Ames test**

Mutagenicity, as well as cancerogenicity, are difficult properties to test in vivo. A significant breakthrough in making mutagenicity test affordable was the creation of cheap and short-term alternatives to the usual rodent bioassay. With this intent, Bruce Ames, at Berkeey University, created a series of genetically engineered *Salmonella Typhimurium* bacterial strains, each strain being sensitive to a specific class of chemical carcinogens (Ames 1984).

The test procedure developed by Bruce Ames was published in 1973 and named after the inventor. The test is extremely fast with respect to rodent studies: it takes around two days instead of years. The test examines whether the chemical added in the test tube to *Salmonellas* creates a reverse mutation in its DNA. Ames test is so known as “Salmonella Typhimurium Reverse Mutation Assay” ; it uses several strains (5 according to most regulations) of bacteria *Salmonella Typhimurium* that carry a mutation in the gene that encodes the amminoacid histidine. This mutant *Salmonella* is know as *His-* and is unable to synthetize histidine, so it is grown in a media with a minimum of histidine.

The chemical to be tested for mutagenicity is added in the media with *His-Salmonella*. If the chemical is mutagenic it can cause a mutation in histidine encoding gene, so His- becomes Hi+ and can again synthetize histidine. Histidine is then removed from the plate, so only the bacteria that mutated (the His+) can survive. After incubation the survived colonies are observed and their number is used to decide whether the chemical is mutagenic or not.

For routine mutagenicity testing, the Salmonella tester strains recommended are five (TA97a, TA98, TA100, TA102 and TA1535.TA1535 is optional, and TA1537 is replaced by TA97a).

To consider the metabolic process in humans, rat liver extract can be added. So the typical Ames test for a chemical requires ten single tests, for each of the five strains with and without metabolism simulation. Metabolism is important as many chemicals become active after being metabolized or vice-versa. Usually it is enough that one test is positive to label the chemical as mutagenic. However, the published results of Ames test in databases usually do not carry a complete list of the ten test values, or they can be based on a lower number of tests.

Using different concentrations of the chemical could produce many more single tests. This is not the case, as it has been observed that the dose-response curve at different concentrations is linear, so indicating that there is no threshold concentration for a substance to be mutagenic. As a consequence for mutagenicity, as well as for carcinogenicity, there is not a safe dose.

As the testing is performed on *Salmonella typhimurium* strains it can provide results not applicable to humans. Some substances such as dioxin gives wrong results in the Ames test, but such substances can cause cancer.

As observed by Ames and explored in other papers (Piergorsh et al 1991) the estimated inter-laboratory reproducibility of this in vitrotest is about 85%. This observation will be taken into account in the conclusive discussion.

Ames test can be applied to many chemicals if they are water soluble. Some benefits of using Ames are the testing procedure is simple, fast, and cheap in comparison with other tests (in Italy the cost of the test for a molecule is about 2000 euros). This cost is affordable for testing a molecule of industrial importance, but it would be too high to test all the chemicals of toxicological concern, or all the molecules generated by combinatorial chemistry in the design phase (millions). Moreover, the test requires that the chemical is synthetized, and this is an extra cost. This explains why the data of Ames test are many, are valuable, and can be used to make predictive models to be used in screening large dataset without the need of synthetizing thee substances.

In conclusion, the results of Ames test for large datasets have a monetary value, contain proprietary information about the molecule formula, and are often proprietary. Most entities are reluctant of making them public. Despite some flaws, Ames test is still one of the most popular tests available. Ames test is mandatory in many regulations, as for pesticides and industrial chemicals both in USA and Europe.

**2.2 Testing Methods for mutagenicity in drug companies**

Ames test is the first step in detecting mutagenicity. For deep screening and in particular during drug development other tests are necessary to evaluate at the cellular level the changes made by chemicals. Those methods are often used with human cells (today it is possible to create them from stem cells) of the organs of interest.

* **High-content screening (HCS).** HCS uses fast automated microscopes and image analysis. It utilizes living cells, which are exposed to different chemicals in different concentrations, takes images, and makes measurements. It is a tool used in many pre clinical drug discovery process, as it allows to see changes in the cells of the relevant tissue.
* **High throughput screening (HTS).** HTS is used to identify modulators of molecular targets, and in general to delineate the relationships between chemical structures and biological activities. It uses automated equipment to test a huge number of samples in a short period of time. It tests compounds at a single concentration, and can work for chemical mixtures, for instance plant extracts. HTS is used in pharmaceutical and biotechnology companies to find chemical compounds with biological activity.
* **quantitative high throughput screening (qHTS).** qHTS differs from HTS in that It allows testing chemical compounds at different degrees of concentrations, so allowing to draw curves are drawn for each chemical compound. It is very popular in toxicology because it provides more interpretability and has lower numbers of false positive and false negatives than traditional HTS. It can work for large chemical libraries (has been tested on more than 105 compounds) to find reliable biological activity.

Ames test is the first test for assessing the mutagenic property of industrial chemicals. If the substance is positive to Ames, then a bunch of other in vitro tests are used to detect whether the effect is in gem cells or in somatic cells. Few experimental data are available for these tests, and it is difficult to develop QSAR models to predict the output.

**2.3 Models for mutagenicity prediction**

For assessing the toxicology properties of chemicals of industrial use, the most convenient way is to collect available test data of the property of concern and make a model. (Quantitative)StructureActivityRelationship - (Q)SAR - are mathematical models that aim to predict biological properties of chemicals (Hansh et al 1962).

QSAR has been created to show the correlation between some chemical features and a property, and are commonly used for drug development and discovery. The roots of QSAR are in mathematical chemistry. In the second half of the past century a rapid progress in developing chemical descriptors and the evolution of pattern recognition and data mining technologies made the development of QSAR models affordable and effective. In short, QSARs are regression models that predict a quantity (a dose).

QSAR is today a mature research field, based on statistical correlations and machine learning, and provides tools for research, regulation, and risk assessment. However it is still a field in evolution, where data checking, chemical descriptors, model development, and result interpretation are still challenging. QSAR is a subset of SAR. QSAR is SAR that models relationship with mathematical equations; SAR is instead based on the observation that specific functional groups in the molecules (called SAs for Structural Alerts) are responsible for the activity; their presence is so used to predict the activity. The two approaches are usually mixed, as many QSAR methods use small substructures as descriptors (fingerprints descriptors), and often the interpretation of QSAR models considers both the role of descriptors and the presence of SAs.

The acceptance of a QSAR depends on its accuracy and interpretability. The reality is that all models, QSAR and SAR included, have some errors, so various tools have been developed to help in assessing whether the prediction can be used or not. Use of applicability domain and local read across can help in this task.

While QSARs were initially developed to model the behaviour of chemicals of the same family, with the implicit idea that the chemicals shared the same mechanism of action, today QSAR models are developed in large chemical spaces, considering that a plethora of new chemicals has been designed, and often adopt non-parametric methods as the ones of machine learning. Neural Networks (NN), in particular, have attracted the interest of QSAR developers as they easily allow constructing effective models, even though their interpretation can be considered much harder than for other methods.

More than 60 models, either free or commercial, are available to predict the output of Ames test. Most of them are built using more or less the same dataset of about 6000 molecules of chemicals of industrial interest and made freely available ((Kazius et al., 2005, Hansen et al. 2009). Many researchers have qualitatively compared the results of those models; a deeper comparison has been made possible in recent years when the National Institute of Health Sciences in Japan made available to participating institutes a set of about 2000 new substances, most of them negative. This dataset has been used as a test set for some freely available mutagenicity models. The results are shortly illustrated in the paper “A large comparison of integrated SAR/QSAR models of the Ames test for mutagenicity” (Benfenati et al. 2019). The study considered 10 models( Table 1) which predict 2 classes (positive or negative) or more classes to account for molecules of dubious classificaton.

**Table I.** The 10 considered mutagenicity models.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model name | Training set | Test set | Chemical descriptors | Algorithm | Predictions |
| Hierarchical clustering (T.E.S.T.) | Hansen | yes | yes | Clustering-  ward method | Pos, Neg\* |
| FDA (T.E.S.T.) | Hansen | yes | yes | Clustering-  Contrera method | Pos, Neg |
| Nearest neighbor (T.E.S.T.) | Hansen | yes | no | Nearest Neighbor (NN) | Pos, Neg |
| CAESAR (VEGA) | Bursi | yes | yes | SVM + SA search | Pos, Neg, Suspect Pos |
| SARpy (VEGA) | Bursi | yes | no | SA search | Pos, Neg, Possible Neg |
| ISS (VEGA) | ISS | no | no | SA search | Pos, Neg, Unknown |
| KNN (VEGA) | Hansen | yes | No | k-NN | Pos, Neg |
| SAm (RHC) | Kazius | yes | No | SA search | Pos, Neg |
| AIm (RHC) | Kazius | yes | No | Modified k-NN | Pos, Neg |
| AZAMES | Hansen  Kazius | yes | yes | RF/CP | Pos, Neg,  Both, Unknown |

A short description of them follows.

* Four models are available in the VEGA platform:

CAESAR (Ferrari and Gini, 2010), based on the Bursi mutagenicity dataset (Kazius et al., 2005), training data set 4204, validation data set 837 compounds. It integrates a SVM model and some expert rules.

SARpy (Ferrari et al., 2013), based on the Bursi mutagenicity dataset, training data set 4204 compounds, validation data set 837 compounds. It automatically extracts SA and uses them as a SAR.

ISS (Benigni, 2008), implementing the rules of ToxTree with a training set of 670 compounds extracted from ToxTree.

KNN using a dataset of 5770 chemicals including a benchmark dataset compiled by Hansen et al. (2009) and a collection of data made available by the Japan Health Ministry within their Ames QSAR project.

* Three models available in the T.E.S.T. platform version 4.2.1 (Martin et al., 2017) of the Environmental Protection Agency (USA): Hierarchical clustering, FDA, Nearest neighbor, all built on a dataset of 5743 chemicals compiled by Hansen et al. (2009).
* Two models available from Nestlè (Switzerland): Structural Alerts model (SAm) (Mazzatorta et al., 2007), and Artificial Intelligence model (AIm) based on an initial version of the Lazar system (Helma et al., 2004), used together.
* One model from Swetox: AZAMES version 2 (Norinder and Boyer, 2017). This model applies the Conformal Prediction (CP) procedure, a new approach for imbalanced datasets.

The comparison of the models on the Japanese test set is reported in Table 2.

**Table 2**. Results of the ten individual models on the set of ~2000 molecules.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Hierarchical | FDA | NN | CAESAR | ISS | SARpy | KNN | SAm | Aim | AZAMES |
| TP | 99 | 138 | 120 | **208** | 187 | 182 | 163 | 178 | 126 | 116 |
| FN | 180 | 172 | 179 | **110** | 131 | 136 | 155 | 140 | 189 | 196 |
| FP | 371 | 467 | 515 | 683 | 592 | 678 | 635 | 377 | 395 | **178** |
| TN | 1532 | 1577 | 1482 | 1426 | 1517 | 1431 | 1474 | 1732 | 1689 | **1925** |
| Tot pred | 2182 | 2354 | 2296 | 2427 | 2427 | 2427 | 2427 | 2427 | 2399 | 2415 |
| Coverage | 0.90 | 0.97 | 0.95 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.99 | 1.00 |
| BACC | 0.58 | 0.61 | 0.57 | 0.67 | 0.65 | 0.63 | 0.61 | **0.69** | 0.61 | 0.64 |
| ACC | 0.75 | 0.73 | 0.70 | 0.67 | 0.70 | 0.66 | 0.67 | 0.79 | 0.76 | **0.85** |
| SE | 0.35 | 0.45 | 0.40 | 0.65 | 0.59 | **0.57** | 0.51 | 0.56 | 0.40 | 0.37 |
| SP | 0.81 | 0.77 | 0.74 | 0.68 | 0.72 | 0.68 | 0.70 | 0.82 | 0.81 | **0.92** |
| MCC | 0.13 | 0.17 | 0.11 | 0.23 | 0.22 | 0.18 | 0.15 | **0.31** | 0.17 | 0.29 |

The parameters considered are reported in Table 3.

**Table 3.** Terms used in the tables presenting the results

|  |  |
| --- | --- |
|  | **Definition** |
| **TP** | True Positive |
| **FN** | False Negative |
| **FP** | False Positive |
| **TN** | True Negative |
| **Tot pred** | Total Predictions |
| **Coverage** | Percentage of predicted molecules |
| **mBACC** | Balanced Accuracy |
| **ACC** | Accuracy |
| **SE** | Sensitivity |
| **SP** | Specificity |
| **MCC** | Matthews Correlation Coefficient |

MCC is a correlation coefficient between observed and predicted binary classifications; it returns a value in the interval (-1, +1), where 1 indicates a perfect prediction, 0 that the prediction is no better than random guessing, and −1 indicates total disagreement. It is computed directly from the confusion matrix as in Equation 1:

\text{MCC} = \frac{ TP \times TN - FP \times FN } {\sqrt{ (TP + FP) ( TP + FN ) ( TN + FP ) ( TN + FN ) } }
 (Equation 1)

This parameter is considered important for unbalanced datasets.

**2.4 Ames test data sets available**

The data used in this thesis have been taken from previous literature or donated.

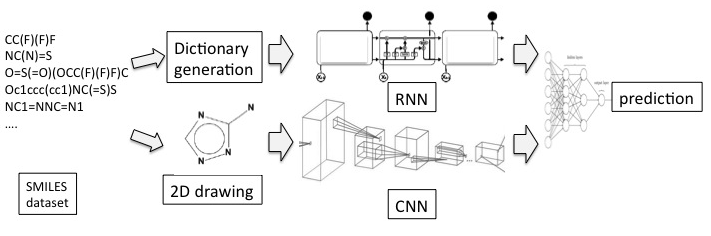
***2.4.1 Zanoli dataset and previous models***

The first dataset has been collected in a previous thesis by Francesco Zanoli. It contains 24003 molecules in SMILES, of which 8127 are positive and 15876 are negative to Ames test. The data set contains proprietary data from the Japanese government.

It has been already used to develop tree deep learning models. The models were developed in two stages: first to create data representation and then to use the extracted features for model building. Two representations are used to train two DNNS: either the picture of the chemical graph obtained from the SMILES (Gini et al. 2020) or directly the SMILES strings (Gini et al. 2019). The reason of using both graphs and SMILES is that their implicit contents can be synergic. The networks have been trained with 80% of the dataset and tested on the remaining molecules.

For input images, a CNN network, called Toxception, is used. The neurons in the input layer receive each a small window of the image; the neurons in the inner layers receive, step after step, smaller windows obtained through convolutions and local averages, so reducing the image to the feature map: from atoms, to bonds, to groups. The features are used in the last layer to create the correlation between the toxicity value and the features.

For SMILES input, SmilesNet is a Recurrent NN which learns grammatical structures within SMILES. A third model is obtained using Toxception and SmilesNet only to automatically extract the features, which are then combined by a third NN, named C-Tox, to make the final classification, as in Figure 2. Table 4 reports the statistics of the three models.



**Fig. 2** The SMILES are used in parallel by SmilesNet and Toxception to extract the features used by C-Tox to produce the prediction

**Table 4**. Performances of the models in terms of MCC, specificity and sensitivity.

|  |  |  |  |
| --- | --- | --- | --- |
|  | MCC | Specificity | Sensitivity |
| Toxception | 0.53 | 0.62 | 0.87 |
| SmilesNet | 0.63 | 0.74 | 0.82 |
| C-Tox | 0.70 | 0.76 | 0.83 |

The results of C-Tox will be compared with the results of the new models.

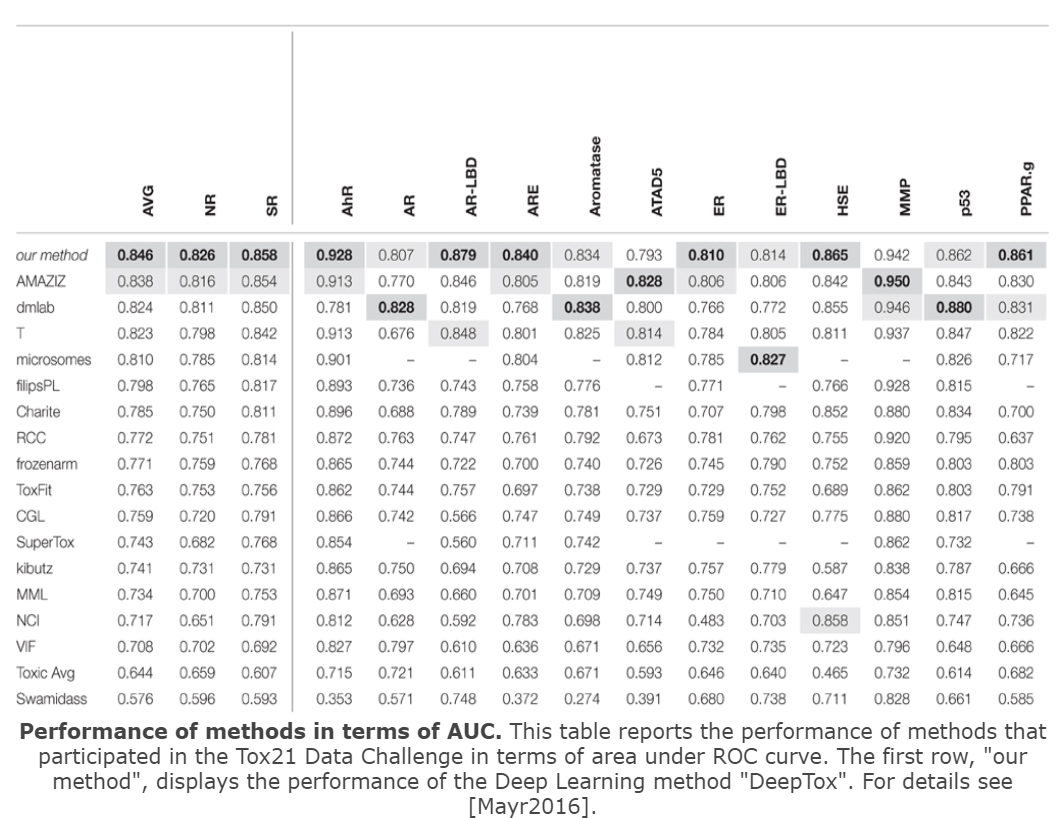
Among the 2000 test molecules, a subset of about 200 has been wrongly predicted by all the tens models. This small dataset will be analyzed further in the new models.

***2.4.2 Japan dataset***

The second data set has been provided by Masamitsu Honma from the Division of Genetics and Mutagenesis in National Institutes of Health Sciences (NIHS), March 2020 for the 2nd AMES / QSAR International Challenge Project. The material consists in: a file with 12134 training chemicals, and a file with 1589 chemicals to be predicted for the challenge. The results of the challenge will be announced in early 2021.

**THIS PART TO GO IN THE NEXT CHAPTER when discussing about DNN - it is not for mutagenicity**

Below is the main model results for Tox 21 challenges. Note our method refer to the DeepTox model that has won the competition but not the models presented in the thesis.



Chapter 3

**Neural Networks for graphs**

**3.1 Neural networks (NN)**

NNs are biologically inspired programming paradigms equivalent to a mathematical function that maps a given input to the desired output. The inspiration of neural networks is our brains, and they emerged from a very popular machine learning algorithm named perceptron by (Minsky, M., Papert, S. 1969).

A NN (Fig 1) is made up of processing nodes, called neurons, which take inputs and convert them to a single neuron output, which is sent to the neurons of the next layer. Weights are learned on the connections to adjust the output of the network iteratively during training. The number of the hidden layer determines the complexity of the function that the network can approximate; nets with one hidden layer can approximate any non-linear function (Chen. T., Chen, H. 1995).

Big networks with many inner layers can approximate more complex functions. However using feedforward fully connected neural nets with many layers is not efficient for many reasons. First of all, determining the number of neurons in each layer is usually done using trial and errors, and the process is very long for more than one hidden layer. The effective use of deep neural networks required to take advantage of the structure of the data to find the net architecture.

Deep learning emerged as a field in machine learning based on neural networks that work as a representation learning (Bengio, Y. Courville, A. Vincent, P. 2013). The most known of such networks, that take the name of deep neura networks, are the recurrent neura networks, the convolutional neural networks, and the graph neura networks.

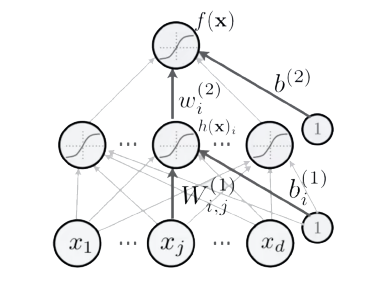


Fig 1 – A schematic representation of a feedforward neural network. Neurons in each layer are fully connected to neurons in the next layer.

In the basic feedforward architecture the network consists of simple processing units, called neurons, and connections, each connection transferring the output of a neuron i to the input of a neuron *j*. Each connection is assigned a weight Wij.

* The lower layer in Figure 1 is called *input layer*, and the neurons in it are called input neurons. No computation is done here within this layer; they just pass the information to the next layer. A weight (usually randomly assigned) is associated with those neurons.
* The top layer, or *output layer*, contains the output neurons, or, as in this case, a single output neuron, which give the result of the network computation.
* The middle layer is called a *hidden layer* since the neurons in this layer are neither inputs nor outputs. The neurons there make intermediate processing, and then transfer the weights to the following layer (another hidden layer or to the output layer).
* h(x) is the *activation function* or transfer function of a node; it defines the output of that node given an input or set of inputs. While a linear activation function was used in the perceptron, only nonlinear activation functions allow neural networks to compute nontrivial functions.
* bis the *bias*. The bias neuron is a special neuron added to each layer in the neural network, which stores an integer value; it has also a weight associated that is learned during training. This makes it possible to move or “translate” the activation function left or right on the graph.

The activation function of a neuron defines the output of that neuron, given an input or set of inputs. Every, or almost every, neuron uses the same activation function. It exists a numerous amount of possible activation functions; they should be, in principle, continuous functions and continuously differentiable.

The first common function described in literature was sigmoid, illustrated in Figure 2.

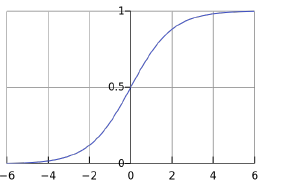


Figure 2 – The sigmoid function

This function is really sensible when the input is near 0; this means that when for very small weights the gradient descent algorithm may incur in the *gradient vanishing problem*: the network cannot improve because its weights are too small and do not affect the output.

Currently the most used activation function is Rectified Linear Unit (ReLU), which is not continuously differentiable but is very easily computed (Figure 3).

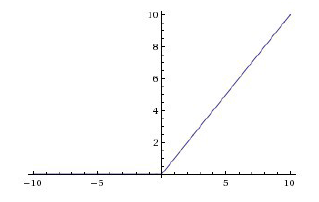


Figure 3 – The ReLU function

**3.1.1 NN learning**

NNs are mainly supervised methods, so are trained using labeled data to approximate the wanted function. During training weights and thresholds are changed so to reduce the error on the training data.

NNs are usually trained using backpropagation of the gradient descent (Werbos, P.J. 1994). Gradient descent is an iterative optimization method that tries to minimize the objective function (error function) by changing the rate of inclination or declination of a slope. It is applied many times using the training data, with the aim at finding the function that fits at best the data. Gradient descent with backpropagation is not guaranteed to find the global minimum of the error function, but only a local minimum.

This algorithm uses the gradient of the loss function with respect to the weights in the NN. The weights updates can be done via stochastic gradient descent using equation 1:

wij(t+1)= wij (t)+ (C/wij )+ (t) (1)

where  is the learning rate, C is the loss function,  (t) is a stochastic term.

The choice of the loss and the activation functions depends on the problem analyzed, and will be reported later on.

After a random initialization of the weights, training data are passed through the net and the output computed. The error (the difference: true value - obtained value) is so computed. The error gradient is then used in backward direction to change the weights using the equation 2:

new\_weight = existing\_weight — learning\_rate \* gradient (2)

During training different loss functions can be used. The most used are Mean Square Error (MSE) and Cross-entropy.

MSE is defined as in equation 3:

(3)

where:

w is the collection of weights of the network,

b is the bias matrix of the network,

n is the total number of training inputs,

a is the vector of outputs from the network when x is input.

Cross-entropy, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from the actual label. Cross entropy is calculated as in equation 4::

(4)

where a is output of the neuron, y is true label and ln is natural log. The cross-entropy is positive, and tends toward zero as the neuron gets better at computing the desired output, *y.*

For training the nets different setting of the hyperparameters are usually tried in order to find the best net. As usually the weights of the net are initialized to random numbers, even repeating the learning with the same hyperparameters can give slightly different weights (it means that another local minimum is found).

* Epochs: one epoch is when the entire dataset is passed once forward and backward through the net. If the data set is big we divide it into batches. As backpropagation is an iterative algorithm, it must be executed for various epochs. A too small number of epochs makes the net to underfit the data, a too big value makes it to overfit the dataset.
* Batch size: it is the number of data present in a batch.
* Iterations: is the number of batches needed to complete one epoch.
* Learning rate: it is a tuning parameter that determines the step size at each iteration while moving toward a minimum of the loss function. It is denoted by the symbol η, and usually it has a low value in (0, 1). η controls how much to change the model in response to the estimated error each time the model weights are updated as in equation 2. Smaller learning rates require more training epochs, given the smaller changes made to the weights, whereas larger learning rates result in rapid changes and require fewer training epochs. Since it influences to what extent newly acquired information overrides old information, it somehow represents the learning speed.

The Optimization algorithms, called optimizers, help to minimize the loss function.

Commonly used optimizers are RMSProp (Root Mean Square Propagation), and SDG (Stochastic Gradient Descent), both using the stochastic gradient descent algorithm to optimize the learning. RMSProp introduces an adapting learning rate for each of the parameters, using the running average of the magnitudes of recent gradients (the first moments) for that weight. Another optimizer is Adam (Adaptive Gradient Algorithm) which uses also the average of the second moments of the gradients (Kingma, D.P., Lei Ba, J. 2017).

Many-layer NNs have been recently introduced to learn highly complex functions. Those networks contain a large number of hidden layers; adding more layers requires a full new way of building the net, so many architectures of deep neural networks (DNN) have been developed, and will be presented later on.

**3.1.2 NN for QSAR modelling**

In QSAR modeling NNs have been used since the ninethies of the last century as algorithms to create the model (Devillers, J. 1996, Gini, G., Katrizky, A. 1999). As in classical QSAR this means assigning to the input neurons the descriptors values. As chemical descriptors are produced in large numbers, usually a selection method is used to select the relevant ones to make the model. See Fig. 4 for the classical way of using a machine learning method to create a QSAR.

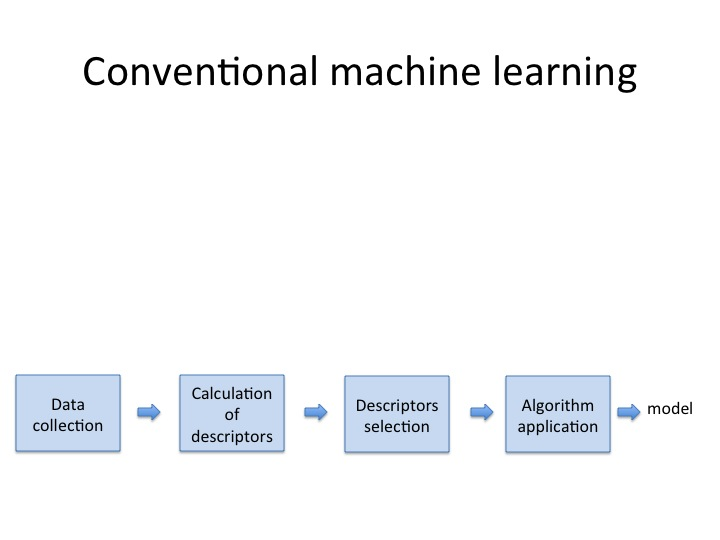


Fig 4 – The steps in creating a QSAR model using machine learning methods

NNs are used with numerical inputs. Chemical descriptors are numbers, whole molecules are instead structured data.

Developing NNs for graph input has been for a long time a research area. Early examples of applications to chemistry are in (Micheli A Sperduti A Starita A 2001). Those methods did not have large imact for two reasons: the network computation was heavy, being based on recurrent neural networks, and the input was humanly crafted. Indeed the usual information about chemical compounds is the chemical graph, which will be presented in the next subsection. The input to the NN was instead a modified set of substructures that required human intervention to be crafted *ad hoc*. Finding NNs able to receive directly the information from public libraries is indeed important. This information is usually stored in various formats, including the 2D structural formula, the SMILES strings, and sometimes the 3D structure. 3D structures are not used in QSAR as they depend on the optimization process use to create them, SMILES are commonly used, 2D structures and their corresponding graphs can be an interesting input.

Some DNNs (LeCun, Y., Bengio, Y., Hinton, G. 2015) that learn directly useful features from data, without the need of computing features, have been developed for chemical applications. Convolutional neural networks (CNN) as well as recurrent neural networks (RNN) have been successful in recognizing images and text respectively, and are now being used in chemoinformatics, especially in drug discovery (Zhang, L. Tan, J., Han, D., Zhu, H. 2017).

All of them use as input the chemical structures expressed as SMILES. It remains to discuss whether an input as a graph is better. Graphs and SMILES contain the same information. However, a possible advantage in using graphs and not SMILES is the lack of unicity in the SMILES notation. This is not a problem when training the net, as it is easy to use the same algorithm to generate canonical SMILES for all the structures, but is can be a problem when using the trained net for a new compound; if the compound is written in a different way it may seen as very distant from the training set and so wrongly predicted. Neural networks accepting a graph representation in input can be of interest; graph neural networks (Wu, Z., Pan, S., Hen, F., Long, G., Zhang, C., Yu, P.S 2019) will so be considered.

**3.2 Chemical graphs as input to neural networks**

A graph is a couple G = <V, E>, where V is a finite set of nodes and E, the set of arches, is a subset of V2. A chemical graph is a model of a chemical system, used to characterize the interactions among its components: atoms, bonds, groups of atoms. In mathematical chemistry a chemical graph, as in Fig. 5, represents the structural formula of a chemical compound.

A chemical graph is an indirected labeled graph, whose vertices correspond to the atoms and edges correspond to the chemical bonds. Its vertices are labeled with the kinds of the corresponding atoms and edges are labeled with the types of bonds.Usually graphs are reduced in size by deleting the hydrogen vertices. The molecular graph normally does not contain any information about the three-dimensional arrangement of the bonds, and therefore cannot distinguish between isomers.

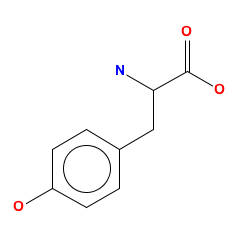
  NC(Cc1ccc(O)cc1)C(=O)O

Fig 5. A chemical structure, its H-depletet chemical graph, its SMILES

H-depleted chemical graphs are the same as the chemical structure without the H atoms. Visiting the chemical graph in top down order produces a string, called SMILES, that is equivalent to the chemical graph (Weininger, M., Weininger, A., Weininger, J.L. 1989). The reverse is not true, as more SMILES correctly represent the same chemical graph. For instance, the aromatic bond of Fig. 5 can be represented by breaking the cycles (as done here) or as a chain of 6 carbons connected with double and single bonds. Another problem is that a small change in the structure can produce a large change in the SMILES; for instance in Figure 6 just one change in the position of a link causes a large change in the SMILES. In this case the similarity of the chemical graphs seems lost in the SMILES.

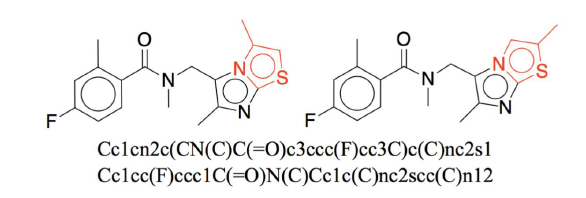


Fig 6 – Two similar molecues and their SMILES

Various researchers have used SMILES as the input to DNN, either as a string (Goh, G., Hodas, N., Siegel, C., Vishnu, A. 2018), Gini, G. Zanoli, F. 2020) or to generate the pictures of the corresponding chemical graphs (Goh, G., Siegel, C., Vishnu, A., Hodas, N.O., Baker, N. 2017, Gini et al 2019). Using chemical graphs directly as input to neworks can present the advantage of reasoning directly on topological similarity.

Chemical graphs are commonly represented as (largely empty) adjacency matrices, as for usual graphs. The adjacency matrix is square; the rows and the columns are the node labels. The entry is used to represent if two nodes share an edge. For the undirected graph, if two nodes share an edge, it is marked as 1 in the entry in the matrix.

Graphs can have numerical or string labels on edges. Numerical labels in graph convolutional networks are called attention; attention uses weights, negative, positive, or null. For zero weights the passing of information of the node from current layer to the next layer is lost. This is called dropout and will be discussed later.

Each node (atom) can have a fixed set of features to describe the characteristics of the node; for instance atom type, number of hydrogen bonds, valence, aromaticity and so on. The feature vaues are inserted in a feature matrix, where the rows are labeled as node Id and the columns are labeled as features.

A node can have many edges attached to it. The number of edges connected to a node is defined as the degree of the node; it can be represented in the degree matrix. The degree matrix is a square matrix, and only the diagonal is filled with values, i.e. the number of edges each node has. The degree matrix is useful in graph convolutional neural networks.

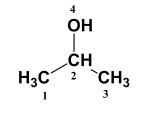
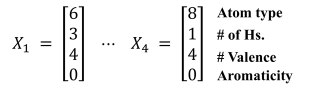
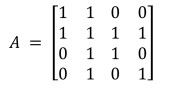
 

Fig 7 – a chemical graph, its adjacency matrix A, the row matrices that constitute the feature matrix.

To use a graph as input for a deep neural network, it seems intuitive to concatenate the adjacency matrix of and the feature matrix, and use the row corresponding to one node each time as an input. However, this is not an efficient approach. There are many problems with this idea.

The first problem is time complexity. The number of parameters in one layer equals the length of the row passed in, which is the number of nodes in the graph+the number of features. Each input neuron has a set of weights connected to the next layer. For a classification task, at least two layers are needed. This increases the number of parameters exponentially.

Another problem is that it does not work with graphs of different sizes. For example, for supervised learning, first let build the model by using each node in the graph as input. It is not possible to train or test on a graph that does not have the same number of nodes as the graph that was previously trained on. The input must be fixed sized. A graph of different size has different dimensions for the adjacency matrix, and this would change the length of the input vector. To train on a larger graph, the dimension of the adjacency would have to be increased to the number of nodes in the graph. It is possible to take the maximum dimension of the graphs and set the adjacency matrix to the number of the nodes in the largest graph. However it is still a waste of computational power and space.

The third problem is nodes are not invariant to ordering. The node can be numbered in many different ways. There are are n\*n-1 ways of numbering the node. Unless the graphs have inherited ordering information, any permutation of numbering on the nodes should provide the same embedding (which means the same input to the neural network.) However, different numbering for the nodes means different ordering of the adjacency matrix. This would change the input for the neural network.

To sum up, there are three reasons against using a concatenated adjacency matrix and feature matrix as input. The approach has a large number of parameters. It does not work for graphs of various sizes. It is affected by how nodes are ordered.

**3.3 Deep neural networks architectures.**

This sub-section introduces the main types of deep neural networks that have been used so far for chemical problems.

**3.3.1 Convolutional NN**

Convolutional neural networks (CNN) are the combination of convolutional layers, pooling layers, fully connected layers and activation layers. The first CNN architecture was developed by LeCun et al. (1995). Its purpose was to classify handwritten digits.

Convolutional layers are characterized by a set of fixed-sized weight matrices called filters. Weights re learned. The filters perform element-wise multiplication on the input image or the output of the previous layer. The filters have the same number of channels as the input map. The output of convolution is a slice fed to the next layer. Each filter contributes one slice of the output volume. The filter has the property of weight sharing. The same filter slides across the entire slice of the input image. In contrast to fully connected layers where different weights are used to connect nodes in between two layers, this creates sparsity of connections and reduces time, space complexity and overfitting due to dense connections.

Spatial invariance is achieved by node aggregation since all the nodes are aggregated without ordering.

Depending on the framework used, the weights can be initialized using different distributions. For Keras, there are “RandomNormal”, “RandomUniform”, “zeros”, etc. In general, the weights are randomly initialized.

Convolutional layers use strides to accelerate the dimension reduction. By default, the filter moves one pixel horizontally or vertically at a time; the stride can be increased to move more than one pixel for reducing the computational cost. The total number of parameters trainable in convolutional layer is: filter-area \*number-of-filters + 1 (for the bias).

Paddings are used for the image map to avoid output shrinkage and losing the edge information. If a smaller output map is undesired, the input map can be expanded by adding zeros to the edges. By manipulating the filter size, the output size can be maintained. Moreover, the filters traverse central pixels more often than the pixels on the edge, so paddings on the edge increases the frequency of the traversal of the filters, thus it can preserve more edge information.

Pooling layers usually come after or in between Convolutional layers. There are two major ways to perform pooling. Average pooling takes the average of the receptive fields. Max pooling picks the largest value in the receptive fields. The choices of hyper parameters affect the operation. Usually no padding is used. Other less known pooling techniques include l2 norm and region of interest pooling. Global Average pooling takes the average of the entire layer for each channel. The output is 1x1xoriginal depth. It can be used to replace fully connected layers because it decreases overfitting. There are no weights to train since pooling uses an aggregation function that would not improve. The choices of using which pooling functions are often empirical. Max pooling extracts distinctive features, whereas average pooling smoothen out the features. Pooling has the benefit of spatial invariance. The object can be shifted a few pixels but still be detected. In general, it reduces the dimensions of the original input, memory footprint and computational cost.

Fully connected layers first flatten the previous layer and connect all nodes of the previous layer to each of its nodes. The total number of parameters is the product of number-of-nodes in the previous layer and this layer, plus one bias for each node of the fully connected layer. Then output is fed into a non-linear layer.

In order to regularize the convolutional layer, batch normalization (Ioffe & Szegedy, 2015) is used instead of dropout. Batch normalization can possibly solve internal covariance shifts. The change in the distributions of layers requires the layers to continuously adapt to the new distribution. The phenomenon where the input distribution to a learning system changes is known as covariate shift (Shimodaira, 2000). Internal Covariate Shift is the change of activation as a result of the change of weights occurred in training. d. Batch normalization performs zero-mean, unit-variance, adds some noise and shifts-scale on each activation. Shift-scale squashes the range of the activation by linear transformation. Batch normalization allows faster convergence since layers become more independent. It allows higher learning rate since it normalizes each layer which prevents the gradients from becoming too small or large during weight update (which are called vanishing and exploding gradients). The weights are easier to initialize. It reduces the need to use drop out since there is already some regularization effect due to the added noise. Since convolutional layers have fewer parameters and are less likely to overfit or have coadaptation, batch normalization is often used instead of drop out right after convolutional layer.

It is difficult to apply CNNs to graphs. Graphs have arbitrary shapes and can not be expressed in pixels.

**3.3.2 Recurrent NN**

There are different ways to handle inputs with variable length, as in case of texts. They can be grouped in two categories. Memoryless models consist of autoregressive models and feedforward neural networks. Memory-enabled models consists of linear dynamical systems, hidden Markov models (Baum 1966) and recurrent neural networks. Autoregressive model uses a mathematical model called weighted moving average for a fixed number of time series data to predict the next value. Hidden Markov models have insufficient memory mechanisms. Feedforward network can be structured by feeding a fixed sized number of time series data to the hidden layers to predict the next value in a supervised manner. Other possible ways to structure it includes using bag of words where the occurrences of different words are counted and recorded in a large vector or using input layers that can fit the largest possible length. However, fixed size window does not model long-term dependencies unless the window size is large enough, but still there is no parameter sharing. Bag of words matches each word from the dictionary, but the order is not maintained. The inability to handle variable length input, long term dependencies, order information and parameter sharing gives rise to using recurrent units with feed forward neural network.

Recurrent neural network (Williams, Ronald J.; Hinton, Geoffrey E.; Rumelhart, David E. 1986) uses the same function and same set of parameters for every time step. At each time step, the previous hidden state and the current input is fed through the function to update the hidden state. The two terms have trainable weight parameters, respectively. The parametrized functions are often fed through nonlinearly. The loss is defined as the sum of the loss from each time step. Once the choice of loss function is decided, the derivatives of the loss can be calculated with respect to the input to hidden state weight at the previous time step, and with respect to the weight for the hidden state to hidden state at the previous time step. To train the model, backpropagation through time (BPTT) is used. It uses chain rules: The first one can be computed by taking the derivative of loss with respect to the output, multiplied by the derivative of the output with respect to the hidden state, multiplied by the derivative of the hidden state with respect to the input weight at the previous time step. The second one can be computed by taking the derivative of the loss with respect to the output, multiplied by the derivative of the hidden state weight. Then SGD???DEFINE THE ACRONYM is performed. The procedure propagates backwards.

S. Hochreiter’s master thesis (Hochreiter 1991) described how during backpropagation, gradients in RNN are inclined to vanish or explode as time length increases. Later formal definition was made available. The exploding gradients is described as the big increase in the norm of the gradient during training because of long-term components growing exponentially larger than the short-term ones. Vanishing gradients is the opposite of exploding gradients where the long-term components grow exponentially to norm 0, which hinders the model from learning correlation between events with wide temporal distance.

BPTT causes either exploding gradients or vanishing gradients. For exploding gradients, gradients can be manipulated in many ways. Some techniques are: L1 or L2 penalty on the recurrent weights can reduce exploding gradient. The regularization restricts hidden state weights to be under 1, which is a sufficient condition to prevent exploding gradients. It has the limitation of not being able to train a generator model or learning long-term dependencies. Teacher forcing (Doya 1993) initializes the model in the right regime and It can be used in training of a generator model or models with unbounded memory lengths. However, it needs a target to be defined at each time step. Echo State Networks (Jaeger and Haas, 2004). does not learn the weights between input to hidden and between hidden to hidden, but only hidden to output. The weights are instead sampled. The gradient becomes just under one, which eliminates explosive gradient but increases chances of vanishing gradient. Gradient clipping (Begnio 2013) is used to scale large gradients down when the gradient goes above a certain threshold, with only drawbacks being necessary to add a threshold parameter.

For vanishing gradients, different techniques are used. Hessian-Free optimizer with structural damping by Sutskever et al. (2011) uses hessian to rescale some components and structural damping to make the model more selective.

Using ReLU for the activation function prevents the derivative from shrinking the gradients when x is larger than zero. The multiplication of many numbers less than 1 results in smaller gradients in early time steps, which makes the weight update ineffective. The derivative of ReLU is set to 1, whereas for tanh is between 0 and 1, and for sigmoid is in between 0 and 0.25. Another way is to initialize the weights to the identity matrix and biases to zero(Hinton, G, Jaitly, N ,Le, Q.2015). This slows down gradients from going to zero. Gradients regularization (Bengio 2013) manipulates the jacobian matrices to keep the changes in gradients small.

Long short term memory (LSTM) was first developed by Hochreiter & Schmidhuber (1997). It is a recurrent neural network with a more complicated linear unit. LSTM uses input gate, forget gate, and output gates in the recurrent unit to control information that gets passed on and stored. There are many variations for LSTM.

Gated recurrent unit (Kyunghyun Cho et al 2014) builds on top of LSTM where the forget gate and input gate are merged, and the cell and output gates are combined ,and added some control flow for old hidden state and new hidden state(using z and 1-z for coefficients.)

Orthogonal initialization (Henaff 2016) initializes weights using orthogonal matrices, which has the property where “all the eigenvalues of an orthogonal matrix have absolute value 11.

Some other variation of RNN includes bidirectional RNN where two directions of next time of the hidden states are used. One goes from old to new and the other goes from new to old.

**3.3.3 Graph convolutional neural networks (GCN)**

Graph convolutional neural networks (GCN) have a more effective way to represent graphs.

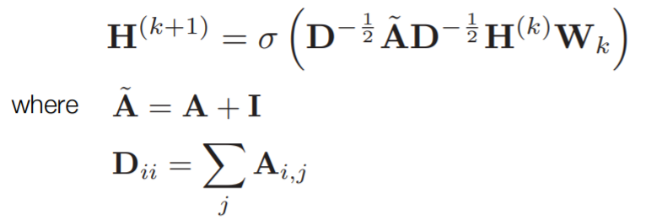
GCN can be either spectral based or spatial based. Spectral-based GCN derives the update function of the hidden states using eigen decomposition of the Laplacian matrix from signal processing point of view, whereas spatial based method derives the update function from neighborhood aggregation which is inspired by CNN. In practice, for GCN, they are the same in terms of implementation.

A spectral-based graph neural network considers a matrix of node features, an adjacency matrix, and the neighborhood aggregation function.

The matrix of the node features uses the node information for features (i.e atom weights) or one hot vector of the nodes if the nodes have no features.

Neighborhood aggregation function aggregates the node features from the neighboring nodes. Different authors have used different aggregation operator, as mean, concatenate, sum, or other. For GCN (Kipf & Welling 2017), the aggregation operator is the sum of the hidden states of the self node and neighboring nodes, divided by sqrt(N(self node)\*N(neighboring node)), where N is the number of nodes neighboring.

The update function of each node at each layer is the output of neighborhood aggregation fed through the same single layer neural network(to allow parameter sharing) with some non linearity added. Equation 5 is used to update the hidden state of the next time step

 (5)

H is the hidden state for the features of a node

K is the current layer

D is the degree matrix

A is the adjacency matrix

W are weights

I is the identity matrix

There are disadvantages for deeply stacked layers. Performing backpropagation of deeply stacked layers requires computing the multiplication of gradients, and this can result in vanishing gradients. After experimenting with different layers (Kipf and Welling) concluded that it is difficult to train the net when there are more than seven GCN layers, as the accuracy gradually decreases as the number of layers increases. A method to mitigate this issue has been proposed in the gated GCN(Li et al., 2016.) where the single layer neural network is replaced by a gated recurrent unit.

There are many ways to optimize the loss function; the typical choice is stochastic gradient descent. The loss function can be selected among cross entropy, hinge loss or ranking loss

Pooling layer converts a graph into sub-graphs to represent higher graph-level representations. It is computationally expensive to use all hidden states from all the nodes for prediction, hence the need to downsample. The downsampling reduces the number of nodes, which reduces overfitting. It also provides spatial invariance, so the nodes ordering does not matter. The pooling operation is usually max, mean or sum. Pooling can be combined with attention mechanisms. Other methods are useing LSTM, which would keep the order dependent information in the graph, or sortpool, which also deals with order dependent graphs. The method is more relevant for directed graph, not for molecules, where the hidden states of all the nodes do not have to be order independent. The least effective method is the eigen-decomposition, used to coarsen graphs according to the topological structure. The problem with this method is it has poor time complexity.

Readout generates graph-level representation by summing up all the hidden states in the graphs after the GCN layer or pooling layers.

The most simplified architecture is: Input graph->GCN->Pooling->Readout->FCN->Non-linearity

There are semi supervised learning, unsupervised learning and supervised learning.

The semi-supervised learning for graph is using a graph consisting of both labeled and unlabeled nodes. From learning from the labeled data, the network would be able to classify the unlabeled data (Kipf & Welling 2017.)

The unsupervised learning is when the graph does not contain labels.The goal becomes learning the function that generates the graph. The algorithm takes advantage of edge level information. This means to predict the label on the edge or connecting strength of the edge. Graph auto encoder (Kipf & Welling 2016) is a typical example of unsupervised learning. The encoder first learns the structure of the graph and outputs the hidden state that represents the graph structure. The decoder reconstruct sthe structure of the graph given the function that generates the graph.

Supervised GCN is done one graph level when the entire graph takes a label and the goal is to predict the label for the graph.

Many different architectures have been proposed:

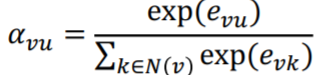
*GraphSAGE* (W. Hamilton, Z. Ying, and J. Leskovec 2017) uses a different hidden state update function. In GCN, to update the hidden state of a node, there are two terms in the function. In GraphSAGE, instead of summing all the hidden states, the neighbor hidden state and the self hidden states are concatenated to preserve more information. Another change is that a generalized operator is used instead of the average operator. This is useful because it becomes useful to analyze different cases based on the theoretical property of different aggregation operators. For instance it is possible to use the pooling operators. The neighborhood messages are first transformed with some strategy. The pooling operator performs element wise mean or max. It is also possible to use the LSTM operator. For LSTM operators, the neighbors have to be randomized since LSTM is not order invariant. To use, it is better to perform LSTM a few times on different order of nodes. Equation 6 is the GraphSAGE hidden state function

 (6)

Graph attention (Velickovic et al.,2018) implicitly assigns the importance ratio to different neighbors of a node by computing the attention coefficient of the pairs of self node and the neighboring node, as in equation 7:

 (7)

The attention weights are normalized to keep all ratios on the same scale with the softmax function (equation 8).

 (8)

Attention weights become the coefficients of the hidden states in the update function (equation 9).

 (9)

The benefits of the attention mechanism are many:

1. it is storage efficient,
2. the complexity of sparse matrix operations is at most O(vertices+edges), which is linear,
3. it has a fixed number of parameters, so it is not dependent on graph size,
4. t is localized since it attends only the neighborhood from the local network,
5. it is computationally efficient because the computation can be parallelized across all edges,
6. it is inductive because it does not depend on global graph structure,
7. the edge mechanism is shared, so it is possible to apply the mode on the new graph.

Multi-head attention (Velickovic et al., 2018) replicates a number of times. Each time a different attention parameter is computed. The replicated attention parameters (which is avu in the figure 8) for each pair of nodes are concatenated or summed.

Oher graph NN are present in literature. (A. Micheli, 2009) presented the earliest work using spatial-based GCN. This model, named NN4G, uses summing for neighborhood aggregation. It also uses skips connection to remember the previous layer. The main difference from GCN is the summing operation instead of taking the mean or taking the mean with the consideration of the degree of the matrix. GCN also uses sum but the neighborhood aggregation and self embedding are concatenated instead of summing them together with different weights.

For really large data, Fast GCNN (J. Chen, T. Ma, and C. Xiao, 2018) uses Monte Carlo integration to approximate the graph data distribution. This is assume the graph is dense enough and does not have many disconnected vertices. If there are disconnected part, the layer does not have any neighbors, it might affect the performance of the model. This way, the computational cost can become much lower. The fast GCNN is great for classification tasks.

In figure 8 there is an overview of different architectures. The differences lie in the aggregation operators, or pooling functions and the operators used in the readout.

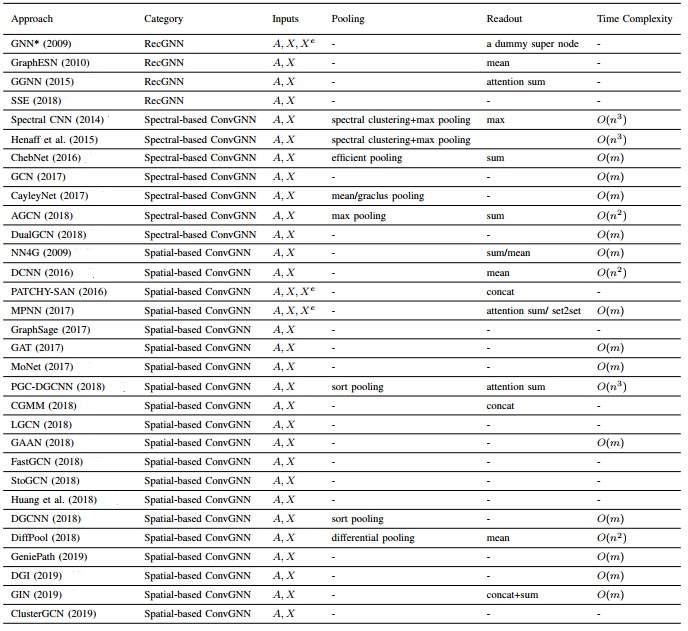


Figure 8 – the different graph NN architectures (taken from….)

**3.4 Bayesian nets and Dropout**

The first known Baynesian neural network (Esther et al, 1989) was built to obtain the generalization error of the network.

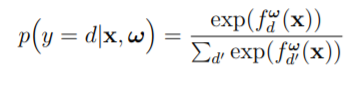
The predictive uncertainty is the co-effect of aleatoric uncertainty and epistemic uncertainty (not algebraically). The predictive uncertainty is the posterior distribution of the Baynesian network or the network output of the classifier after applying the sigmoid function for binary classes or softmax for multi-class. To find the predictive probability of the output, it is important to find the posterior probability of the weights first. To find the posterior probability of the weights (which is the probability after the model has seen the training data) Bayes rule is required. The posterior probability over parameters after seeing the data = the probabilistic function by which the weight generate the output in the training given corresponding input(known as likelihood), multiplied by the prior probability over a parameter, divided by the probability distribution of the target data given its corresponded input (known as evidence). Once defining the prior and the likelihood, the posterior probability over weights can be calculated.

How to define the prior:

The prior distribution over the space of weights in the neural network describes how likely for each weight to generate the target output(y) in the absence of the training data(x,y). Once the classifier has seen the data, the probability will change the probabilities on weights that have greater contribution to the output. There are many different priors, but many priors are inappropriate for Bayesian settings. The most popular choice is Gaussian prior. Gaussian prior has zero mean and it will lead to integral with closed-form solution (trackable.) Another benefit of using Gaussian priors is in practice optimizing the model weights by adding sum of square weights (known as L2) regularization to minimize the difference between the true value and the predicted value (known as loss) is equivalent to setting Gaussian priors.

How to define the likelihood:

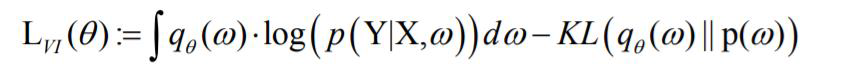
This is the probabilistic function by which the inputs generate the outputs given the weight distribution. Gaussian likelihood is a common choice for regression, and softmax likelihood is for classification. It takes an exponent of one output divided by the sum of exponents of all the outputs, as in equation 10.

 (10)

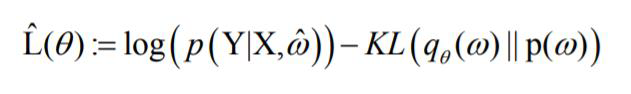
Once the posterior probability distribution over weights is computed, the posterior probability distribution over the prediction outcome given the training set by integrating it with respect to weights. The integration is the likelihood multiplied by probability of the weights given the training data set. However, in reality, the distribution of the weights given the training data might not have a closed form solution because there is no closed form solution for the distribution of data. The distribution of data requires integration but has to be done with respect to the weights,which can be multi dimensional and thus very computational expensive.There are many different techniques to work around this problem. For instance, Monte Carlo sampling incorporates a similar distribution and then takes the finite sum to replace the integral.

However, this can be computationally expensive in deep networks. Variational inference is another workaround for this problem: instead of finding the posterior distribution of the weights given the training data, it finds another distribution that is similar to analyze and closest to the posterior distribution.

In order to find this (variational) distribution, Kullback-Leibler divergence (Kullback & Leiber 1951) is used. The Kullback–Leibler divergence calculates the dissimilarity between two probability distributions. It is always non negative. Two probability distributions are the same if the Kullback–Leibler divergence is zero. KL is non-symmetric. Reordering the arguments in the equation will lead to a completely different formula. To minimize the dissimilarity of two distributions, this means to minimize the KL function with respect to the weights. This is done by adjusting the weights to make the alternative distribution fit closely to the original distribution. The formula is equivalent to maximizing the lower bound of the log likelihood of the probability of the training target given the corresponded input(also known as evidence). The equation can be reorganized via Jensen’s equation. Jensen's equality tells us the log of the mean >= mean of the log. The equation becomes maximizing the lower bound of the evidence. The weights that maximize this objective is the weight that makes this alternative distribution q a good approximation for the posterior distribution. The loss formula is in equation 11.

 (11)

Notice that this is an integration that has no closed form solution (or formulas to simplify it for calculation). To approximate this integral, Monte Carlo integration can be used. The integral is replaced by log of evidence and the formula is converted to one step loss formula of equation 12:

 (12)

If we change the sign, the first term is loss. The second term is the sum of squares (for weights) regularization. Maximizing the above equation is the same as minimizing the negative of the above equation.

In each step, 1) a random set of averaged weights in each layer is put in the objective function 2) minimize the objective function. This process is repeated many times. The more the better, because the more stochasticity (randomness) added to the network, the less sure the model is on its prediction. This gives a higher and more accurate uncertainty.

In practice randomly sampling the weights from the layers is the same as randomly setting the nodes in the layers to zero, which is what drop out does. Minimizing the objective function derived from KL is the same as minimizing the loss function, which is the cross-entropy function in practice.

Monte Carlo dropout (Y. Gal and Z. Ghahramani 2016) finds the predictive mean but still can't find the exact variational distribution. It uses the concept of variational distribution by sampling from a similar distribution in order to calculate the means because there is no way to calculate the mean for the original distribution because it is too complicated to work with. In practice in order to find the similar distribution, it is the same as evaluating the loss function+weights regularization. The purpose of using monte carlo drop out because it works like monte carlo integration. We can just sample a few instead of approximating the whole parameter space. To calculate the uncertainties, we just need the mean and variance sampled in the layers.

This is different from the traditional drop out. Traditionally dropout is used mainly to reduce overfitting. In the training phase, dropout layers are used, but in the test phase, dropout layers are removed and weight scaling is used to make adjustments of the test data for the weights trained.

The reason to use Bernoulli distribution is the following: the variational distribution q is decided to resemble dropout. In each layer, q is defined as the Bernoulli distribution that takes in the mean of the weights of that layer, multiplied by the diagonal matrix of the Bernoulli variable(which consists 1 for some probability, 0 for the others). In other words, some weights of each layer are set to zero. This is equivalent to using dropouts.

Some other reasons for using Bernoulli's are that it is computationally cheap to have multiple models, that it has fixed variance (the posterior probability uses variance of the distribution), and the variance of the weights is the square of the mean of (weight\* variance of Bernoulli distribution)- which is probability of 1s\*probability of 0s.) The variance for Bernoulli distribution is fixed, which means the weights have to decrease when the data increases. The strongest regularization of the weights is 0.5 for dropout.

In Bayesian recurrent neural network, the parameters ( which are the hidden-to-hidden state weights, input-to-hidden state weights and the bias) used in the function to update the hidden state of next time step are converted to random variables with Gaussian priors. The parameterized function is plugged in the log likelihood function. The rest of the procedures to compute the posterior distribution of the weights can be referred to the explanation in the Bayesian neural network. For LSTM, the weights for all four gates (cell, input, forget, and output)) are converted to stochastic variables with Gaussian priors and the procedure is the same as above.

Many papers discuss the technique to tune the drop out rate. Concrete Dropout is a special form of dropout developed by (K.Alex, Hron.J, Gal. Y 2017). The dropout rate tuned by hand or grid search usually takes a lot of time because each time using a different drop out requires to perform an experiment. Usually the model has to run for some time, and then it becomes possible to compare the performance of models using different dropouts. A technique is proposed by (Y. Gal and Z. Ghahramani 2016) called concrete dropout. Concrete dropout find the optimal dropout rate using standard gradient descent technique to find it. For Bayesian neural network, the dropout probability characterizes the posterior uncertainty, so tuning is important. The method substitutes the Bernoulli variation distribution at training time with a gumbel-softmax/concrete distribution. This enable some techniques to optimize the gradient of the evidence lower bound with respect to the dropout probabilities.

**3.5 Performance parameters**

Different types of Uncertainties are considered:

* Aleatoric uncertainty. Aleatoric comes from the latin word aleator that means dice player. This uncertainty comes from the inherent noise in the data, which is a result of imprecise measurement. This leads to class overlap(meaning positive points appearing in negative zones or vice versa.) It is also irreducible uncertainty since by increasing more knowledge on the dice rolls, the randomness of the dice roll can still not be reduced. However aleatoric uncertainties can be reduced by improving the experiments that performed the measurements of the data.
* Epistemic uncertainty. Epistemic comes from the Greek word episteme, which means knowledge. Epistemic uncertainty is the knowledge uncertainty. This is the result of the classifier’s lack of knowledge on the test data. The test data can be very far away from the distribution of the training data (which means it is completely outside the domain of the classifier) or slightly far from the distribution( which happens when the classifier has not observed enough data from that distribution.) Epistemic uncertainty can be reduced by acquiring more knowledge on the data, so by adding more training data.
* Systematic uncertainty: This is the uncertainty that can be reduced.

Most of the performance metrics for 2-class classifiers are based on the count of true positive (TP), true negative (TN), false positive (FP), and false negative (FN).

About the performance of classifiers, the following measures are considered:

Accuracy

is not a reliable metric for the real performance of a classifier, because it will yield misleading results if the data set is unbalanced. For this reason additional metrics are more important.

Sensitivity (or recall, or positive rate) is the proportion of real positive cases that are correctly predicted; this is defined as:

Specificity (or true negative rate), is defined as:

Precision (or positive predictive value) is defined as:

The F1 score is the harmonic mean of precision and sensitivity:

The Matthews correlation coefficient (MCC) measures the quality of binary classifications. It is generally regarded as a balanced measure, which can be used even if the classes are not comparable in size. The MCC is in essence a correlation coefficient between the observed and predicted binary classifications; it returns a value between -1 and +1. A coefficient of +1 represents a perfect prediction, 0 means no better than random prediction and -1 indicates total disagreement between prediction and observation.

**Chapter 4.**

**Design of new models for mutagenicity**

As described in the previous Chapters, the mutagenicity property is of great practical use and models to predict it are still challenging.

The aim of this thesis is that of providing new models according to the following targets:

* create models using the chemical graph
* create models with improved interpretability and uncertainty measures;
* create a new model using the biggest available Ames data set (i.e. Zanoli data set);
* create a new model more oriented to industrial chemicals, using the Japan NIHS data set;
* participate to the Japanese contest and predict their test data set.

In the following the choices made for the creation of the models are illustrated.

For both data sets a couple of models is built. The first model uses GCNN, the second adds the Bayesian interpretation as proposed in the paper “A Bayesian graph convolutional network for reliable prediction of molecular properties with uncertainty quantification”(K., Woo, K., Yongchan, R., Seongok 2019).

**4.1 The data sets**

For the Ames models, the training data consists of 80%(or 19202 SMILES) of the Ames data set (=24003 entries); the other 20% is used for the testing set. A small percentage of the training set is used for validation. Additional tests are performed on a small set of about 200 molecules that are wrongly predicted by all the available mutagenicity programs, to check in which measure they are correctly predicted. It is presumed that this set is out of the domain of the Ames data set, or that it contains wrong experimental values.

For the NIHS models, the dataset from National Institutes of Health Sciences (NIHS) of Japan is used. The training data consists of 12140 chemicals from NIHS and Hansen open data. Each consists of a serial id, ANEI No., registration number, the chemical abstracts registry service (CAS) registry number, chemical name, structure, molecular weight, formula, and ANEI phase, and most importantly the Ames result. The Ames result is classified with four categories: strongly mutagenic, mutagenic, non-mutagenic, Non-applicable (NA.) Some results are reviewed by the experts and reassigned to NA for reasons such as ambiguous result, unorthodox protocol, etc. For binary classification purpose, only two labels are used: NA is eliminated, and strong mutagenic and mutagenic are classified as mutagenic.

For the QSAR challenge only the NIHS model will be used to classify the test set of 1589 SMILES given by NIHS. The test data set contains SMILES, a serial ID, the registration number, CAS registry number, and the chemical name. The CAS number is a unique identifier for a chemical compound. It avoids the confusion with different names assigned to the same chemical. According to the challenge rules is not possible to use other data not provided by NIHS.

**4.2 Choices for the GCNN network**

The basic network architecture considered is presented in Figure 4.1. It is a GCNN with….

**ADD here figure and short description.**

The other choices about input, optimization, training are illustrated in the following.

***Network input***

In the Ames data set the longest SMILES contains 503 characters. This means the largest graph has less than 503 nodes, but there are many SMILES that have much fewer nodes. Therefore, the graphs do not have enough nodes to use deeper layers.

The atom features are represented as one hot vector in the feature matrix. The atom feature used are H, C, O, N, and all the other symbols are regarded the same. Other chemical symbols such as Br, F, etc are not used since they are not the major constituent of known structural alerts. NOT CLEAR- WHY? ALL THE ATOMS PRESENT IN THE DATA SET SHOULD BE CONSIDERED

The SMILES symbols such as ‘(’, ‘)’,’[‘, etc.. are not included in the feature set since the adjacency matrix already provides the relative positioning of the atoms. The adjacency matrix is generated by Rdkit library (REFERENCE) from the SMILES.

For example: CC(CC)C is represented as in Figure 4.2.

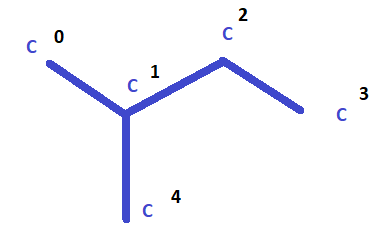
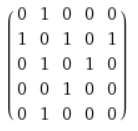


Fig 4.2. The graph representation of the SMILES string CC(CC)C

The adjacency matrix is thus 5 by 5 and represented as in the equation 4.1:

 (4.1)

To generate the hidden state of the node, since the feature matrix is 2D, it goes through 1D convolutional neural network to extract higher level features. The filters are set at a smaller dimension than original dimensions of the feature matrix. The kernel and bias weights are initialized using Xavier initialization. The resultant feature matrix is the hidden state of the node.

***Attention layer***

The main criteria to consider for choosing the most suitable approach are the prediction performance and time complexity. Attention mechanism has led to higher performance and so it has been selected.

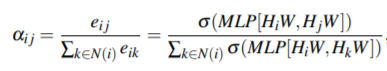
For the attention layer, many different attention score formulas have been considered.

*- Additive score function* (Bahdanau 2015). The function calculates the score using the hidden states of a pair of nodes and the weight associated with the pair. The activation used is hyperbolic tangent. In practice attention score is a N x N matrix, where N is the number of nodes in the graph. It is used to multiply the adjacency matrix to produce a weighted adjacency matrix. The weighted adjacency matrix is multiplied by the NxF feature matrix, where F is the number of features for each node. The result is the weighted neighbourhood aggregation of all the nodes, as in Equation 4.2.

 (4.2)

The attention weights are initialized using Xavier initializer. Xavier initializer initializes the weights to be zero mean and 1/n\_input to make the variance of the input and output to be one. It helps avoiding vanishing and exploding gradients.

*- Softmax.* Another popular choice is using softmax (Luong2015) illustrated in Equation 4.3. It takes in the hidden states of a pair of nodes and passes in the single layer neural network and use softmax to normalize the scores of different pairs of nodes. However since the attention layer is used to extract structural alerts, normalization of the local neighbourhood of a node may affect finding them.

 (4.3)

*- Attention weights.* Multi-attention mechanism uses a different attention weight for each computation of the attention score for each pair of nodes. The hidden state of a node is computed using different attention weights each time. Then each variation of the hidden state is concatenated and passed in a 1D convolutional neural network to produce a single hidden state.

- *Node aggregation*. To update the hidden state of a node, there are several ways to perform neighbourhood aggregation, as proposed by (Kipf, T. & Welling, M. 2017). A basic approach is to average all the neighbour’s hidden states and add the old hidden state, and pass them to a non-linear single layer multi-layer perceptron (MLP) to learn the weights. In this model, the hidden states of the nodes are updated using attention-weighted sum of the neighbourhood. The non linear layer used is ReLu since it offers better performance and also can possibly decrease vanishing gradients.

In the readout phase, all the nodes in the graph are aggregated using their respective hidden state. The aggregation method used is the sum operator, which is a standard method used among majority of the architectures for graph classification. It preserves the linear relationship of all the nodes in the graph and maintain order invariance of the nodes.

Other available methods include averaging, max pooling or concatenating the hidden states from all the nodes.

* With averaging, the sum of the hidden states is divided by the number of nodes in the graph. One drawback is the classifier cannot distinguish molecules that have the same ratio of atoms. For example, Holmium’s molecular formula is Ho. Hydrogen peroxide’s molecular formula is H2O2. Two chemicals are completely different but are classified the same with average pooling.
* It is possible to use max pooling even though a lot of information of the graph would be lost. Another problem is if two molecules both contain the same atom that has the maximum value, but the quantities are different. For example, O2 and O3 will be classified the same since the largest atom is O and average pooling would also fail since the average does not change.
* Concatenating all the nodes for graph classification can eliminate the problems with average and max pooling but in practice some dimensional reduction technique should be applied to reduce the computational cost. Concatenation is used in Graph isomorphism network (CITATION?).

Other algorithms such as graph coarsening are not ideal since they are difficult to implement and there is no performance guarantee.

THIS FIGURE IS NOT DESCRIBED. CUT?

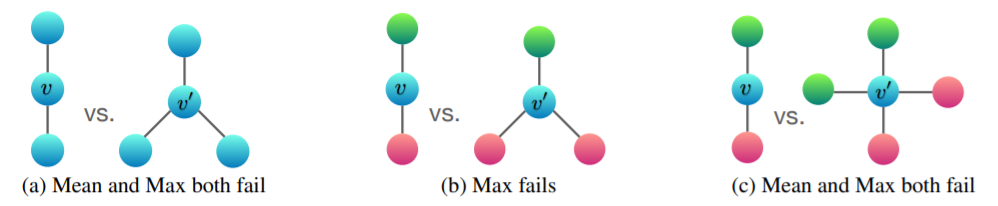


Fig . taken from how powerful are graph neural networks ?? SOURCE? IT S NECESSARY?

**4.3 Choices for training**

***Optimization***

Early stopping is used to stop the training process when the validation accuracy stops improving for 5 epochs. The validation set is helpful in determining the proper hyperparameters. Cross validation is not used because of the additional computational cost.

Leave one out is a validation technique. For each epoch, it averages the validation results of all the data points. It uses a validation set of one data point by calculating the difference between the true value and the predicted value of this data point, put this data point back in the training set, take another datapoint to validate till all the data points are used, then the difference between the true value and the target value  of each data is averaged. The averaged result is the validation loss for one epoch. The benefit is leave one out is almost unbiased since it averages the validation results, so it can serve as early stopping. However, it comes with high computational cost. For large dataset, as our Ames data set, the computation becomes very costly because it is performed number-of-data\*(number-of-data -1) times.

K fold cross validation splits the data in k parts. It is similar to leave one out except each time it takes one part out of k parts of the data and calculates the validation error. It trains the rest of the data(which is the k-1 parts) and calculates the validation error on each part till all the parts have been validated exactly once. The validation loss is the averaged validation loss of each fold. The benefits of k fold cross validation is, when k is properly selected, a lower computational cost than leave one out, and as a result can be used on large dataset. k fold cross validation detects overfitting, so it could be used as a way to stop training when the validation loss stops decreasing for a period of time. However, our data is highly imbalanced, there are many negative samples but very few positive samples. This could create a problem. When the model is trained on k-1 fold the model can have high specificity. This means the model can not detect positive samples. The validation result can be misleading because there can be many folds/parts of the data that does not contain the minority class(which is the positive class).

Stratified k fold cross validation, where each fold contains the same ratio of positive labels, has to be considered instead. However, it is not used because of added computational cost.

Holdout validation is the most basic form of validation among leave one out and k fold cross validation. Hold out validation splits the training data in subsets of a percentage desired. Then it performs validation on such a dataset. It calculates the averaged difference of the true target value and prediction value of each data point. It helps with early stopping and hyperparameter selection.

For hyper parameter selection, different models can be created by using different hyper parameters such as changing the filter size of the convolutional layers, adding non linearity, etc.. The validation loss of different models can be compared and find the model with the best selection of hyper parameters.

In conclusion, hold out validation is preferred to other validation techniques for the following reasons:

* For leave one out, since there 19203 SMILES, for just calculating validation loss alone, it has to calculate 3652278006 times+ the average operation. This is very computationally expensive, and it would take too much time.
* For k fold cross validation, since the data is highly imbalanced, it is possible that the model is trained mostly on non mutagenic folds of the data and validation of most folds may have low validation loss. The result is the validation accuracy would be much higher than the test accuracy. Even though for the purpose of hyper parameter selection, k fold cross validation is still effective. The point of using it becomes comparing the performance of each model created by different sets of hyper parameters. All the models have high validation accuracy, so the task becomes a matter of finding the best set of hyper parameters.

However, for the purpose of hyper parameter selection it is not necessary to use leave one out, a representative hold out set is enough to help deciding the best hyper parameters.

To find a representative enough hold out set it is important to reduce the correlation between SMILES. The Ames datasets has many (about 2000) identical molecules represented by different SMILES, sometimes clustered together in sequential order. If there are many of those duplicates in the validation set, the validation might not be representative enough. Low validation accuracy can be seen as just having data distribution far from the training data. To not let such factors influence the interpretation of the validation accuracy, randomization of the dataset is necessary. To reduce it, the data is randomly shuffled. This eliminates data dependency.

To solve the problems induced by the imbalanced dataset, the percentage of positives in the training set and validation set are kept the same. This solves the problem where the training consists of mostly negative, but the validation set consists of mostly positive, and vice versa. In the scenario, the model would fail in predicting the positive since it has not seen enough positives. There is another concern regarding find the right size of the validation set, if the validation set is too small; it may be not representative enough for early stopping.

All the optimization problems will be solved using the Adam optimizer (REFERENCE).

**4.4 The Graph Bayesian Architecture**

The two data sets used to create the GCNN models are also used to create the Bayesiam models.

Bayesian Deep Learning indicates a theoretical justification for DropOut using Bayesian principles. Using DropOut at inference time and not just during training allows one to estimate the uncertainty of a trained model. It has been introduced by (Yarin Gal et al. 2016) along with a list of references). In practice what this method does is to get the uncertainty inherent in the data, but is not using Bayesianpriors. So in practice it is a method for measuring some combination between the noise in the dataset and the noise in the network training method. As it is easy computable it has been seleted for improving the interpretation of our models.

For Bayesian graph convolutional neural network models, drop out layers are added. Dropout layer is placed after every hidden layer to make the model able to quantify uncertainties. The drop out layer should have the same drop out ratio for each drop out layer. Often finding the optimal drop out ratio takes trial and error. It can be very computational expensive to perform different experiments. To find the most accurate drop out ratio, concrete dropout (K.Alex, Hron.J, Gal. Y 2017) is used. It finds the optimal dropout ratio using its in-built algorithm.

For prediction, the model predicts the same batch of data for a number of times and the results are concatenated and the concatenated results are passed in a sigmoid function and they are averaged. The averaged prediction value is then fed into the equations to calculate aleatoric uncertainty and epistemic uncertainty.

**4.5 Model interpretation and Feature Extraction**

The structural alerts are extracted using attention mechanism where a specific threshold is set to generate the substring. The threshold determines how long the substring is. One way is to sum all the attention scores for each node from the attention layer (as seen in fig 4.3), and then use maximum continuous substring. This way the substring with highest score can be found.

Another way is to use a fully connected layer before the readout layer to perform weighted sum on all the nodes, and extract the weights, but in practice it has led to poor prediction results.

Adjacency matrix (batchx503x503)

Feature matrix( batchx503x30)

Convolutional layer

Output: batchx503x64

Weighted adjacency matrix

Batchx503x503

Attention Layer

Output:batchx503x64

Feature extraction

Output: batchx503x1

Attention Layer

Output:batchx503x64

Readout Layer

Output: batchx1x256

Fully connected Layer

Output:batchx1

Fig 4.3. Full architecture with feature extraction.

**Chapter 5**

**Results of the GCN models**

This chapter reports the experiments and the results of using GCN on the two data sets Ames (Zanoli) and Japan. The two best models are reported.

**5.1 Architecture choices and training**

The architecture choices are similar for both the models.

**5. 1. 1 Data**

The two data sets are available as tables, as already indicated; another column has been added to indicate the chemical class to be use to make statistics.

There are no universal definitions of the chemical classes, and in particular of organic chemicals. For instance the Agency for Toxic Substances and Disease Registry (ATSDR) to address hazardous substances (<https://www.atsdr.cdc.gov/substances/ToxChemicalClasses.asp>) considers grouping chemicals according to chemical structure (e.g. hydrocarbons), uses (e.g. pestcides), physical properties. The chemical classes of interest for QSAR are:

* Benzidines / aromatic amines
* Dioxins, furans, PCBs (phenyl ring of carbon atoms)
* Halogenated
* Hydrocarbons
* Nitrosamines /ethers alcohools
* Organoposphate and carbamates
* Phenols /phenoxy acids
* Phthalates

Various systems exist to classify chemicals; all of them use some nix of the functional and chemical properties; moreover classes are overlapping as one molecule often can be classified in different chemical classes. For our purposes a rough indication of the chemical class is important to make statistics about the models so to indicate to regulators which chemical classes are better or worse predicted. If the molecule under prediction is chemically classified, the predicted value can be weighted with the prediction accuracy of the chemical class.

The classes considered in our data sets have been obtained…HOW? With SARpy? It is impossible

They are represented into a hierarchy; as for many classes there are too few molecules, only the first hierarchy of classes has been considered.

ADD figure or table about the chemical classes USED

For Ames data, the training set is composed of 80% of the data set of 24003 SMILES. The validation set is 0.01% of the training data. The test set is 20%, which corresponds to 4800 SMILES. The data are randomized with seed 0 using numpy.

For Japanese competition, the training set is composed of 12140 SMILES from Masamitsu Honma. The data are randomized with seed 0 using numpy. The test set is 1598 SMILES.

Since it is important for the model to not have any predefined knowledge about structural alerts, only the chemical information is represented. The recognizable feature set includes all types of atoms seen in the training set. Each atom is represented by one hot vector. Additional chemical information of an atom added in the features are degree of atom, total number of hydrogen, implicit valence, formal charge and whether the atom is aromatic.

**5.1.2 Hyper parameters**

The set of hyper parameters used include different hidden dimensions, max atoms, number of layers, batch size, epoch size, learning rate, regularization scale, pooling strategy. Different configurations of hyper parameters have been experienced to achieve the optimal performance of the model.

* The max atom size is configured to be the largest possible SMILES in both training and testing set. This is needed because the atom size corresponds to the dimension of the adjacency matrix and feature matrix. The SMILES will be skipped if the atom size is too small.
* The number of layers is set at 2 to reduce the computational expense. From the experiments, setting at a higher number of layers increases the computational time drastically. The model uses gated mechanism to boost the prediction accuracy. In the experiment, the optimal is at two. In the attention layer, multi attention is used. Different number of attention heads are considered. The attention layer computes the hidden state of each node using weighted neighborhood aggregation. Attention head is most ideal at 2. Attention heads higher than 4 require higher computation time; attention head equal to 1 results in slightly worse prediction accuracy.
* Batch size has shown to have significant change in the results for some models. In the experiments batch normalization with small batch tends to make the training unstable, so batch normalization is eliminated. The optimal batch size for Ames model is 512 and for Japan model is 128 based on experimental results.
* For learning rate, Adam optimizer automatically tunes the learning rate for each parameter. In the paper, it is recommend to set the learning rate at 0.01, and in our model it has shown that a learning rate higher than 0.01 make the model learn too fast and possibly only converge to local optimum. Although a learning too small might also cause overfitting. However, since Adam optimizer automatically tunes the learning rate, we set the learning to be default at a small value.
* Batch normalization can be used to regularize the convolutional layer. The convolutional layer produces the hidden state of each node. Since graph convolutional network tends to overfit more where training accuracy is much higher than test accuracy, as seen in the experiments. Batch normalization is standard way to regularize convolutional layer and it outperforms drop out layers. However, in the experiment batch normalization does not improve overfitting. The difference between training accuracy and test accuracy remains large. In fact, the training accuracy converges slower but there is no improvements for test accuracy.
* Max pooling is often used to improve the prediction accuracy. In the experiment, max pooling can be combined with the sum of all the hidden states. During readout, the node with the maximum hidden state plus the sum of all the hidden states. If max pooling is used instead of sum pooling, this can lead to problem with graph isomorphism (discussed in chapter 4) where different SMILES are recognized the same by the model. However, there could be interesting information generated using max pooling. Therefore, max pooling is combined with sum pooling. The prediction performance improve slightly with such combination. The activation function uses either sigmoid or ReLU. The differences are not much in the experiment. Minimum pooling is not a standard pooling strategy and it also offers lower performances.

**5.2 Results and statistics of the models**

**5.2.1 GCN Ames model**

For Ames data, batch of 512 offers much better performances than all the other size of batch. The model oscillates slightly at first but from epoch 10, the model slowly improves till over 90%. The training accuracy is only slightly than the testing accuracy.

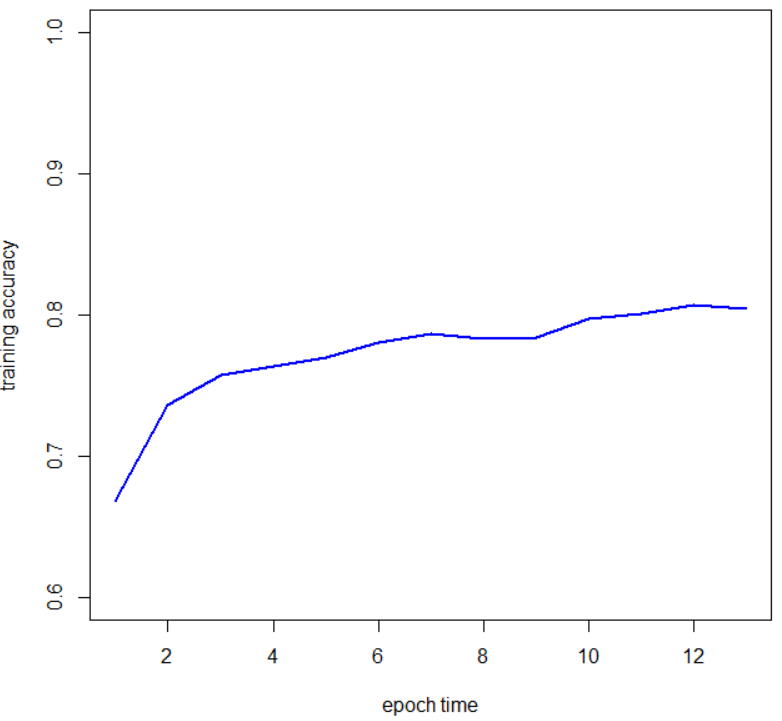
For batch of 1024 the computation time is much longer. The difference between training accuracies and test accuracies are lower compared to other batch choices. This is contrary to many current architectures, where small batch adds more noise to the model and thus provide regularization effects at small learning rate. In our experiment, larger batches actually decreases overfitting. Other factors can be taken account. It is possible for the learning rate, certain batch size provides better generalization. From the trend of the graph, it is hard to tell if the model would eventually as training goes on since there are some oscillation at the beginning. However, the experiment is not continued since the time and space complexity is quite high.

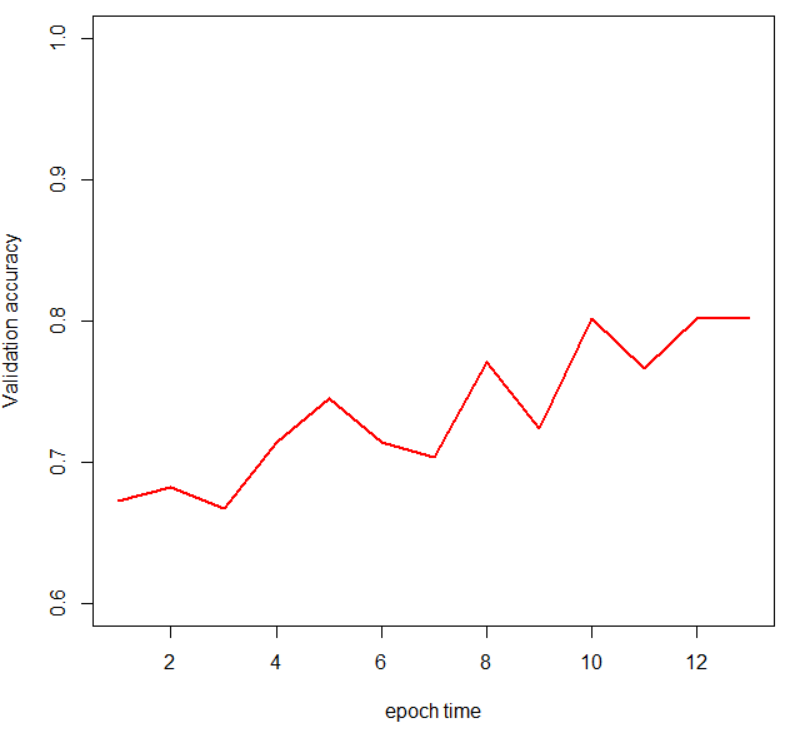
The optimal model uses attention mechanism in the readout layer (as seen in the figure 1.) GCN tends to achieve the optimal prediction performance at the first few epochs and the improvement slows down as the number of epochs increase. The computation for each epoch is 37 minutes.

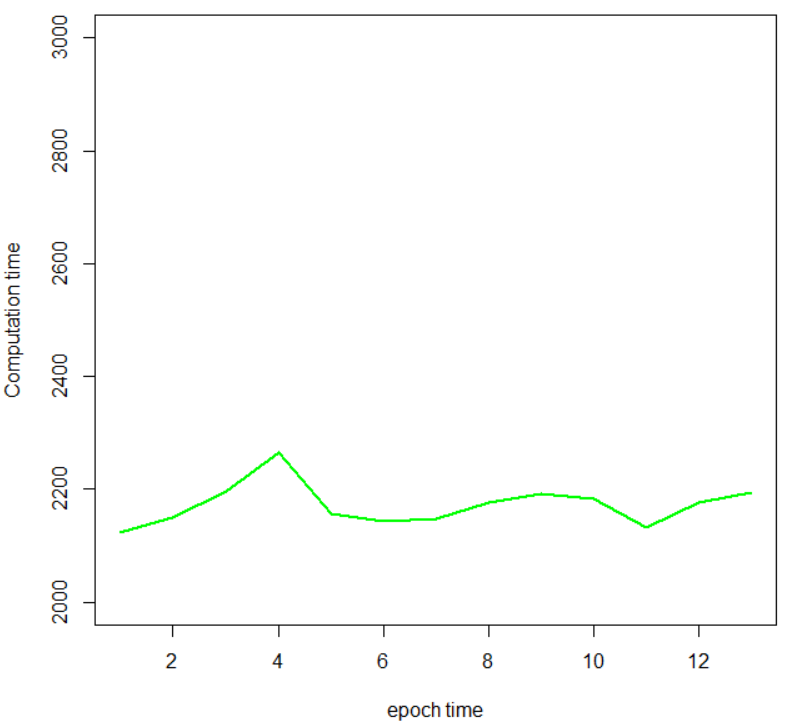
For hidden states of the node, the hidden states are computed using weighted neighborhood aggregation and the result is passed to a convolutional layer. The weighted neighborhood aggregation uses multi attention where for each attention weight, an hidden state is computed. To produce the final hidden state, all the outputs are averaged.

The architecture uses additional weights to add importance to each node. Since the model does not use bidirectional LSTM, there is no backward direction for the hidden state. Therefore, the additive attention score function can not be used without some modification of the architecture. The model instead uses both the hidden state of the nodes and the target state, which uses a dummy node as to link all the nodes, to feed into the fully connected layer, and uses hyperbolic tansion as the activation function. The output of the fully connected layer has the shape of (max atom size x 1). Thus a score of 1 unit is generated for each node. To normalize among the nodes and to have scores add up to be one, softmax is used. The softmax layer is followed by two dense layers. The attention weights performs weighted sum of all the nodes. The first has 256 neurons, and the second one has one neuron since it is used for binary classification.

HERE ADD THE FINAL RESULTS AS STATISTIS AND STATISTICS PER CHEMICAL CLASS







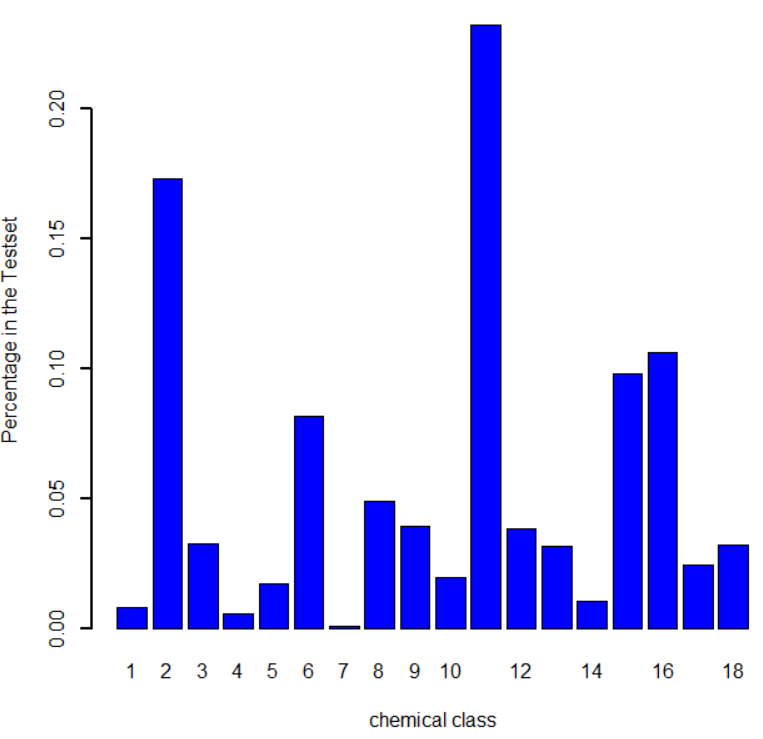
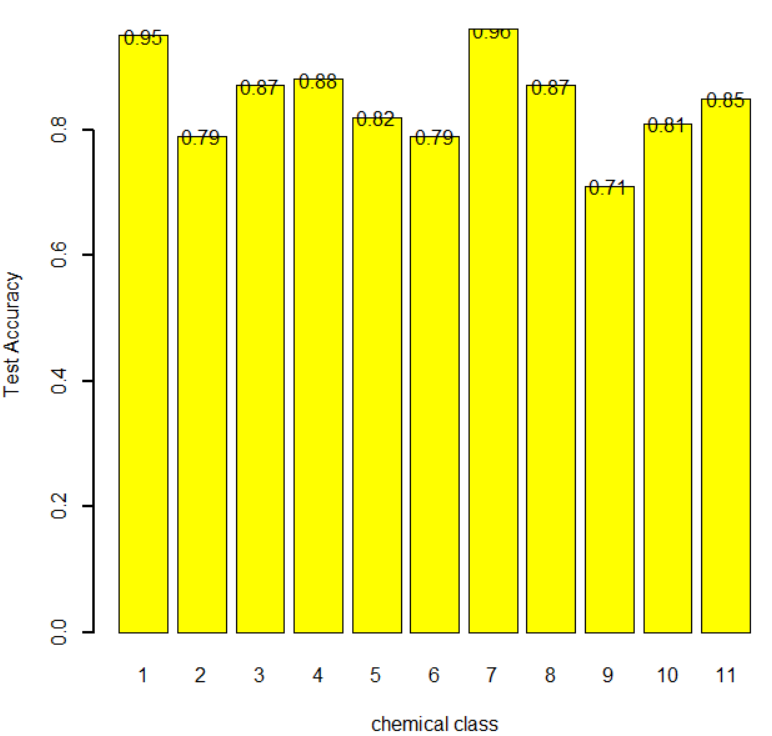
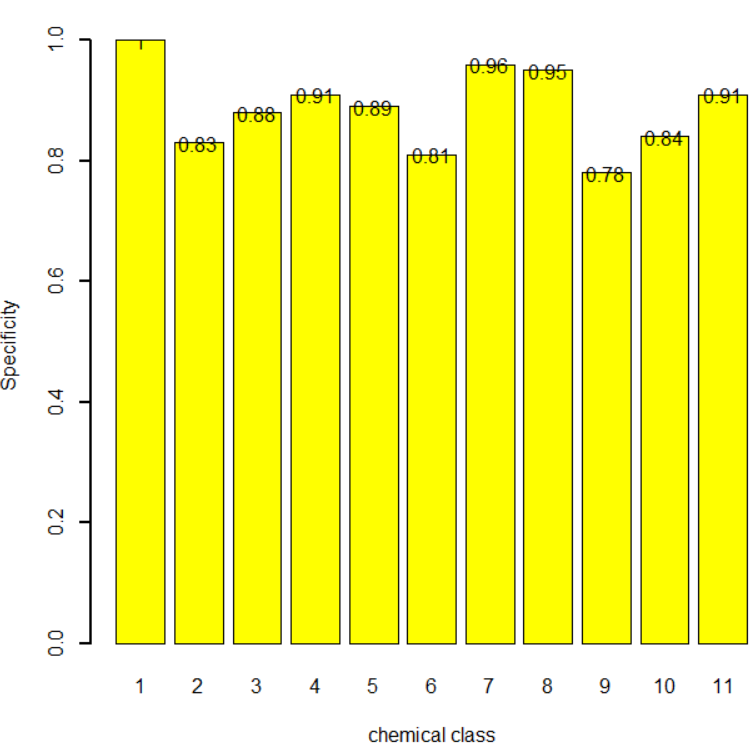
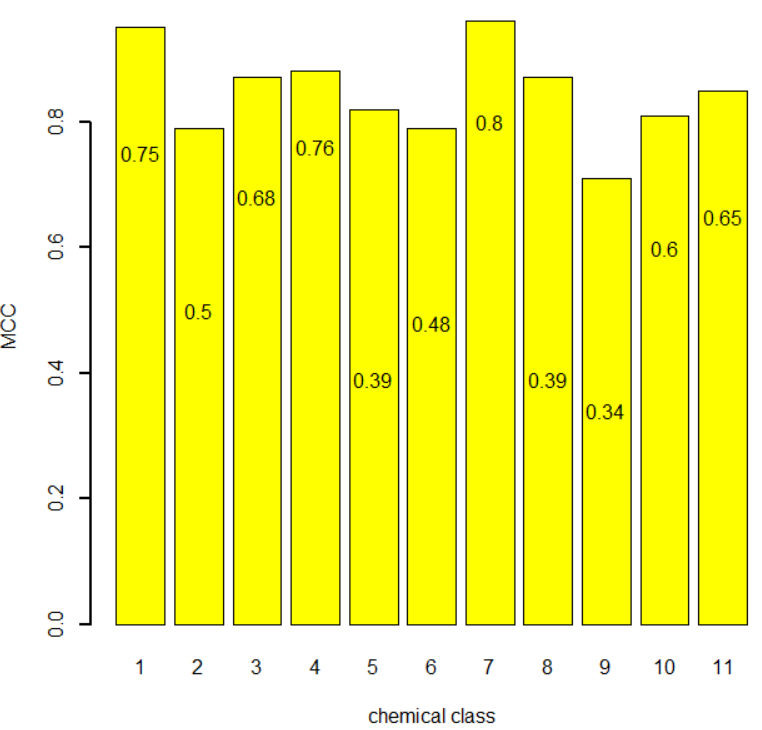
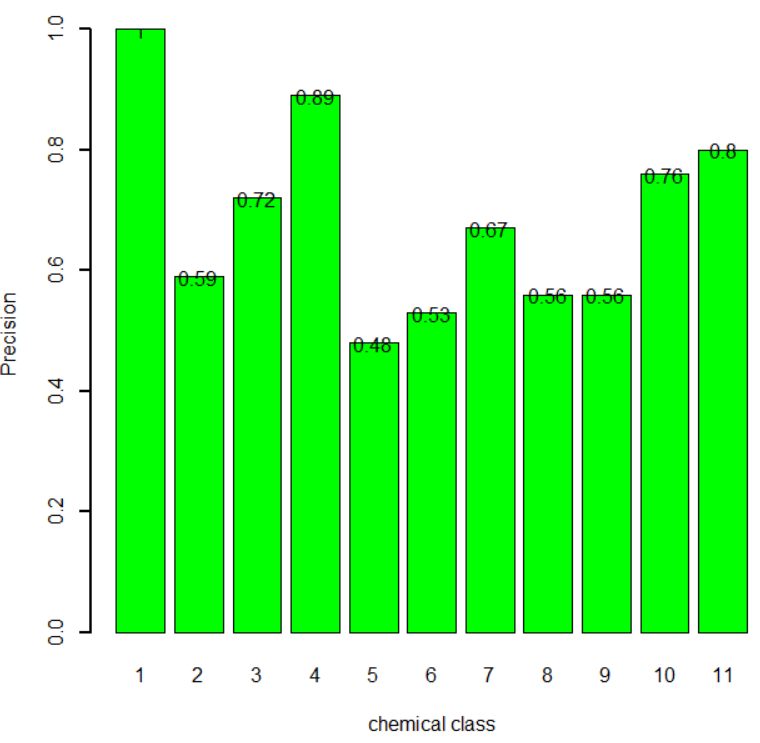


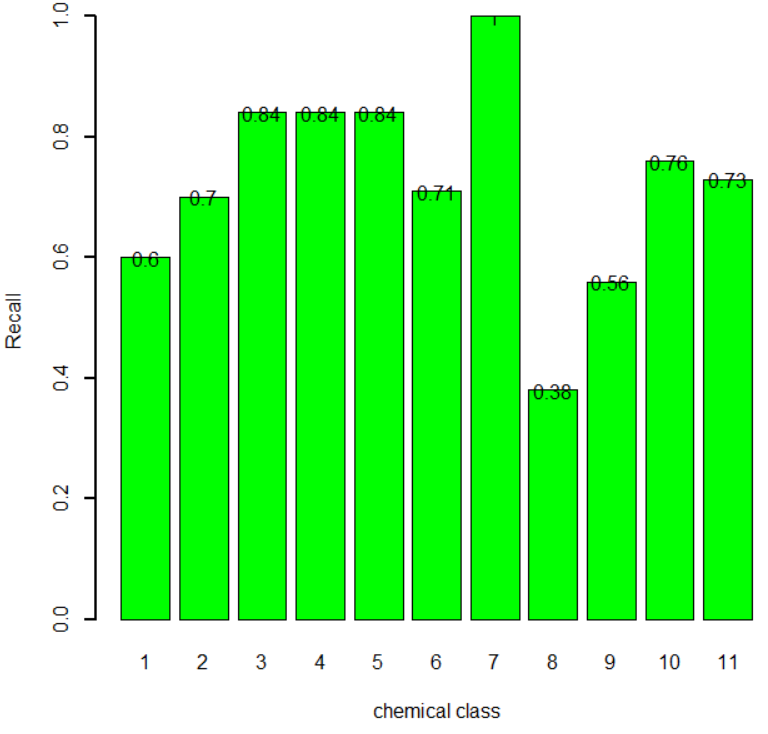
Fig. 1.Alcohols and polyols, 2. Benzene and substituted derivatives, 3. Benzoic acids, 4. Hydrocarbons, 5. Biphenyls and derivatives, 6. carboxylic acids and derivatives , 7. Organic thiophosphoric acids, 8. Ethers 9. Lipids and lipid-like molecules , 10 Organohalogen compounds, 11. Organoheterocyclic , 12. Organonitrogen compounds , 13. Phenylpropanoids and polyketides ,14. phenols , 15. Others , 16.Other Benzenoids ,17.Other Organic acids and derivatives ,18.Other Organic oxygen compounds

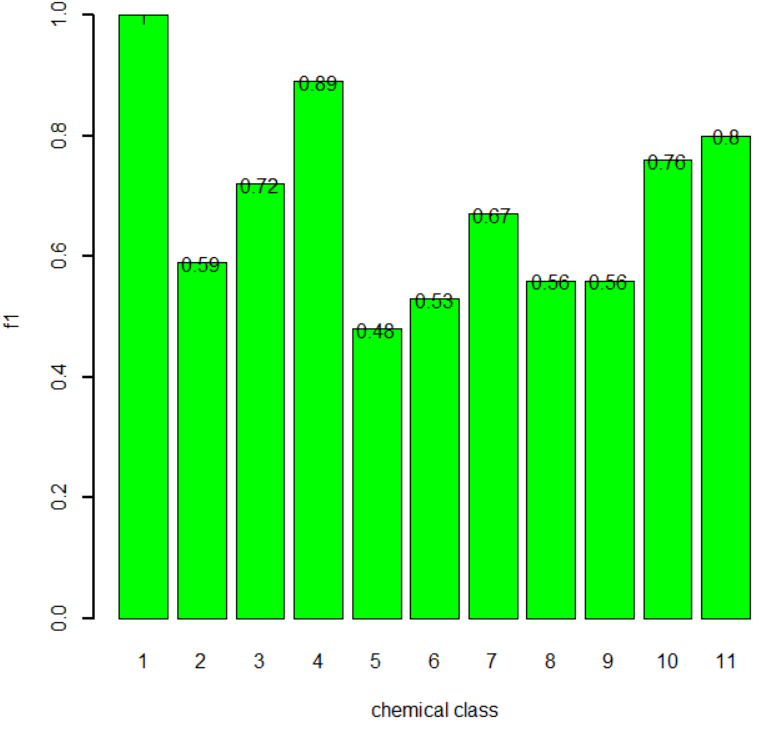












Add models comparision

Results are also computed according to the chemical classes. 21 chemical classes are considered. SMILES in the same chemical classes usually have similar structure. Thus there could be a common structural alert that can be extracted in the same class. The extraction of the structural alerts will be presented in 5.3.

The net is illustrated in Figure 1.

**5.2.2 GCN Japan model**

For Japanese competition the optimal batch size found is 128. The optimal model uses attention mechanism in the readout layer (as seen in the figure 2.)

The architecture uses a different attention score function. It is a similar to the Ames model but uses a simpler attention function. The model instead uses only the hidden state of the nodes to feed into the fully connected layer, and uses hyperbolic tangent as the activation function. The output of the fully connected layer has the shape of (max atom size x 1). Thus a score of 1 unit is generated for each node. To normalize among the nodes and to have scores add up to be one, softmax is used. The softmax layer is followed by two dense layers. The first has 256 neurons, and the second one has one neuron since it is used for binary classification.

The net is illustrated in Figure 2.

HERE ADD THE FINAL RESULTS AS STATISTIS AND STATISTICS PER CHEMICAL CLASS

**Gate Coefficient:**

Fully connected layer

Nonlinearity(Sigmoid)

Output:batch X feature

Fully connected layer

Batch X 503 X64

f

A:Batch X 503 X 503

H:Batch X N X F

Convolutional layer

Output:batch X 503 X 64 hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh

Node aggregation

Output:batch X feature

Nonlinearity(Relu)

Output:batch X feature

Convolutional layer

Output:batch X 256

Attention

Attention weight

concatenate

Z

1-Z

Batch X 503 X feature

Fully connected layer

Output:batch X 503 X 1

Nonlinearity(tanh)

Output:batch X 503 X 1

Softmax

Output:batch X 503

Flatten

Output:batch X 503

Reduced\_Sum

Output:batch X feature

Nonlinearity(Sigmoid)

Output:batch X feature

Fully connected layer

Output:batch X 256

Readout

Fully connected layer

Output:batch X 1

Round(>0.5)

Fig 1. Japan model.

Batch X 503 X feature

Fully connected layer

Output:batch X 503 X 1

Nonlinearity(tanh)

Output:batch X 503 X 1

Softmax

Output:batch X 503

Flatten

Output:batch X 503

Reduced\_Sum

Output:batch X feature

Nonlinearity(Sigmoid)

Output:batch X feature

Fully connected layer

Output:batch X 256

Readout

Batch X 503 X feature

Fully connected layer

Output:batch X 503 X 1

Nonlinearity(leaky relu)

Output:batch X 503 X 1

Softmax

Output:batch X 503

Flatten

Output:batch X 503

Reduced\_Sum

Output:batch X feature

Nonlinearity(Sigmoid)

Output:batch X feature

Fully connected layer

Output:batch X 256

Readout

Concatenate

Output:batchX504X62

Random normal(batchX1X62)

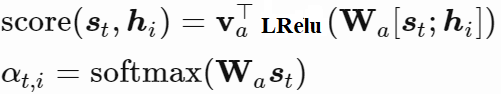
Fig 2. Ames model.

**5.3 Feature extraction**

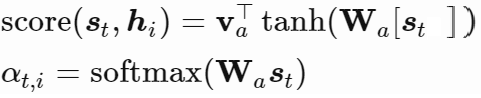
In readout layers, all the nodes are summed up to produce the graph feature. However this approach does not allow a meaningful interpretation of the importance of the substructure since the aggregation of the nodes is not weighted. Since the model is not built using seq2seq model (REFERENCE), attention is performed in three ways.

Calculating the attention strength of each adjacent pair of the atoms and summing all the connection strength for each atom. The larger the aggregated attention values, the more important the substructures are.

A different method is to perform attention at the readout phase. At readout, all the nodes are aggregated, the idea is to perform attention on the aggregated hidden state and the hidden state of each atom. Then pass the attention matrix to the fully connected layer. More specifically, the readout layer is modified to use a dummy node as the target hidden state to link all the nodes, and the nodes in the graph as source hidden state to pass in a dense layer with leaky relu to calculate the attention weights. Using softmax to normalize them and then using the attention weights to perform weighted sum of all the nodes.



The third method is to use only the hidden state of the source state which are hidden state of all the nodes in the graph and use a tanh as the activation function. It only passes the source hidden state to the dense layer with tanh.



For every substructure extracted, the program matches each substring with the 24003 Ames data and checks if the polarity of the structural alerts matches the experimental value of the SMILES. The structural alerts are validated by comparing each postive or negative structural alert with the SMILES containing the alerts. THIS IS NOT CLEAR

Then the statistical results are produced.

More details about the SAs re discussed in Chapter 7 that considers the interpretability of the models.

Chapter 6.

Bayesian Graph Convolutional Neural Network

**Chapter 9.**

**Conclusions**

In this research, a new model for mutagenicity is developed. The model is graph convolutional neural network and Bayesian graph convolutional neural network(GCN). Bayesian graph convolutional neural network allows for uncertainty estimation. Uncertainty estimation tells the researchers the if the model has enough knowledge of predicting the data. They are built using the largest possible dataset: Ames dataset. Another two models are built on the Japanese competition data. The results provides statistical results on different chemical classes to tell which are better predicted. The previous literature( Gini et al. 2019) uses smiles string so it can not visualize the structural alert in the graph since it has lot important details of the graph. There is advancement in model interpretation for this work. Structural alerts can be extracted and visualized in 2D automatically. To validate the structural alerts, they have been compared with the known structural alerts and shown nearly 0.7 accuracy. The model has predictive accuracy of 0.8 which is in line with previous state of the art. The prediction for Japanese competition test data are made to participate in the competition. The model has shown to converge fast and reach 0.8 test accuracy at 12 epochs but with long computation time(=37 minutes) for each epoch due to large batch size.

In chapter two, mutagenicity and related concepts are defined. Different experiments for mutagenicity are described in detail. Different testing methods for drug company are introduced. There is some brief review of current best mutagenicity models and models from previous literature. The datasets for Ames data and Japanese data are described.

In chapter three, there are description of neural network and its learning. Application of neural network in QSAR is explained. Deep neural network, recurrent net, graph convolutional neural network, Bayesian neural network and regularization techniques are described. The performance parameters used for our models are described.

In chapter four, there is description of the datasets selected and their motivation: making use of the largest public data, participating to the Japanese challenge, providing more interpretability of the model by adding uncertainty estimation. The process of converting the data to neural network input are described. There are explanation of different layers, different choices of optimization, hardware and library used. The architecture of the GCN model used in this thesis are described.

In chapter five, the GCN architecture and hyper parameters are described in detail. The results of training and testing are provided. The performance of using different hyper parameters and different configuration of the architecture are provided in graphs. The statistics of each chemical class for numerous performance parameters are included.

In chapter six, the complete Bayesian GCN architecture and hyper parameters are described in detail. The results of training and testing are provided. The performance of using different hyper parameters and different configuration of the architecture are provided in graphs. The statistics of each chemical class for numerous performance parameters are included. The uncertainty estimation are added to the statistical results.

In chapter 7, there is the discussion of the interpretability of the model. The comparison of the structural alerts with expert knowledge such as Toxtee are given. The visualization of the structural alerts are provided in 2D. There are discussion of the importance of interpretability of the uncertainty and how it affects the prediction of the model. There are discussion of the significance of the model architecture, and hyperparameters. . The performance of different chemical classes are provided with interpretation.

**Future work**

There can be some possible future improvements for the 201 wrongly classified data to make the model has better ability to generalize data that has distribution far from the Ames data. Other improvements can be made to improve the comparability of the structural alerts extracted with the known structural alerts. This can be done by improving the overall model predictive performance since the predictive performance is in proportion to the quality of the structural alerts extracted.