

# Massively Multitask Deep Learning for Drug Discovery

By  
Jason Feriante

A PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE IN COMPUTER SCIENCES

AT THE

UNIVERSITY OF WISCONSIN-MADISON

2015

---

ADVISOR SIGNATURES:

ANTHONY GITTER

JUDE SHAVLIK

## TABLE OF CONTENTS

Abstract .....	1
Acknowledgements .....	1
Introduction .....	1
Bioinformatics: History .....	2
Bioinformatics: Practical Applications .....	3
Bioinformatics: Experimental Approaches .....	4
High Throughput Screening (HTS) .....	4
Artificial Enrichment: Docking .....	4
Artificial Enrichment: Machine Learning .....	4
Artificial Intelligence: History .....	5
Machine Learning: Introduction .....	7
Artificial Enrichment: Machine Learning .....	8
Fuzzy Expert Systems .....	8
Genetic Algorithms .....	9
Inductive Logic Programming .....	10
Logistic Regression .....	10
Decision Trees .....	11
Random Forests .....	12
Multitask Learning .....	13
Neural Networks: The Perceptron .....	14
Artificial Neural Networks .....	15
Deep Learning .....	16
Deep Learning: Optimization .....	17
Neural Networks: Deep Belief Networks .....	18
Neural Networks: Deep Autoencoders .....	20
Experimental Evaluation .....	21
The Datasets .....	21
Training, Validation and Test Sets .....	23
Experiment Design .....	24
Pyramidal Deep Belief Networks: Multi and Single Task .....	25
Experiment Results .....	25
Achieving Higher Performance With DBNs .....	26
Conclusions .....	27
References .....	28

## Abstract

The University of Wisconsin-Madison Small Molecule Screening Facility (SMSF) performs research to discover compounds that bind with specified targets (usually proteins). Due to time and cost constraints, the SMSF is limited to real screening on a very small pool of candidate compounds with high throughput screening (HTS) methods. Our research team proposed to enhance or replace current virtual enrichment methods with massively multitask deep learning to substantially increase the odds of HTS success. Our challenge was to first create a working model and then to extend that model to provide an ongoing predictive solution for the SMSF. During this summer term project the first phase was completed, which involved the replication of past experiments with deep learning models and other baseline methods. This project report describes my research efforts and experimental results.

## Acknowledgements

During the course of this project I had the opportunity to work with some great academic minds: Professor Anthony Gitter (Ph.D), Spencer Ericksen (Ph.D), and fellow researcher Vaidhyanathan Venkiteswaran (“Vee”). Thank you Professor Anthony Gitter for your brilliant, and patient guidance as well as your support in making this project a reality. Thank you Vee for recruiting me to work on this project, as well as your many creative and insightful solutions you have come up with in response to many of the difficult problems we faced. And to Spencer Ericksen (Ph.D), thank you for helping us understand the problem domain as well as your substantial work in collecting all the various required datasets and in generating the of ECP4 fingerprints. And finally, to Professor Jude Shavlik (Ph. D) thank you for your assistance in reviewing this project.

## Introduction

The University of Wisconsin-Madison Small Molecule Screening Facility (SMSF) is tasked with drug discovery, which requires the identification of molecules that bind non-promiscuously with specified protein targets. The lab faces increasing cost challenges as the ratio of drugs discovered annually per billion dollars spent has declined by nearly 50% per year since the 1950s [2]. While large pharmaceutical firms such as Merck and Pfizer continue to spend a fortune on brute force high-throughput screening (HTS), this is not a viable option for the SMSF and many other academic screening facilities.

Due to time and cost constraints, the SMSF is limited to screening a relatively small pool of candidate compounds with HTS methods. Among the millions of molecules that are potentially viable and safe for humans, only a few are in fact good candidates for any given target protein. Due to the systemic problems with traditional drug discovery, new methods must be found to increase the declining efficiency of traditional approaches. One possible solution to this problem is virtual screening for increased enrichment levels. Enrichment refers to increasing the density of good targets in a pool of candidates as opposed to a random selection baseline.

“*In silico*” (Latin for “in silicon” [49]) refers to running a simulation on silicon semiconductors as opposed to running lab experiments. With *in silico* screening, computer simulations are used to prioritize targets for HTS based on predicted binding interactions. Virtual screens can be one of two types: (i) structure-based and (ii) ligand-based. The SMSF team currently uses a structure-based virtual screening process that leverages multiple virtual docking-based algorithms (AutoDock and AutoDock Vina). Using this process, the team is able to achieve an enrichment in the range of 5 to 10 times better than random selection. Unfortunately, these enrichment levels are still too low to obtain several high-quality hits in a screen of 100 to 1000 small molecules. Higher enrichment levels are required to have a significant impact on the screening workflow.

Recent work performed by researchers at Stanford and Google shows great promise in multitask, deep learning based methods of virtual screening in “*Massively Multitask Networks for Drug Discovery*” (MMNT) [1]. Our research team proposed to enhance the SMSF's virtual enrichment methods with similar multitask deep learning methods. Our goal was to first replicate and then to extend previous work to provide a working solution for the SMSF. The scope of this summer 2015 project is the first phase of this research and includes: data collection, cleaning, and the replication of the machine learning models used in the MMNT experiments on our own data.

This report includes general bioinformatics and machine learning to provide context, as well as our experiments, results and conclusions. The project report is organized as follows: 1. bioinformatics, 2. machine learning, 3. experiments and 4. conclusions. The sections on bioinformatics and machine learning provide context for the project and outline the foundational concepts on which our experiments were built. Various historical methods in each discipline are explored and contrasted with more recent approaches to show the motivation for our decision to use a multitask, deep learning ensemble for drug discovery.

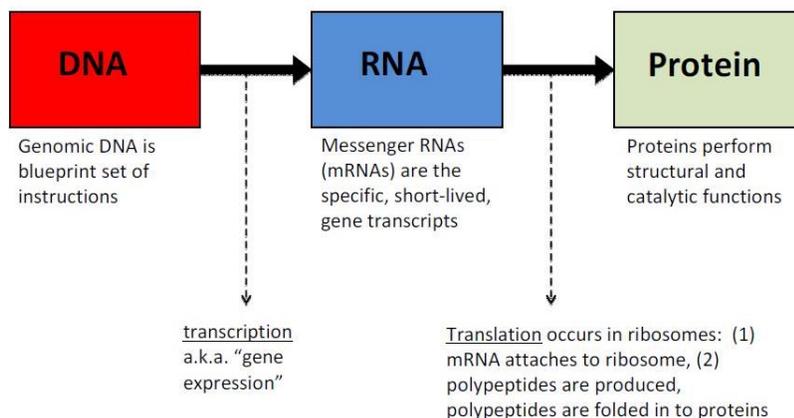
## Bioinformatics

As biological information continues to grow exponentially with data-driven scientific advances such as the Human Genome Project, the necessity for computer-aided analysis has grown increasingly important. Bioinformatics has evolved to combine several other fields including: computer science, statistics, and applied math. One goal of bioinformatics is to store and analyze biological data for the purpose of drug discovery and development.

### Bioinformatics: History

All living organisms depend on three primary biopolymers: proteins, Deoxyribonucleic acid (DNA), and Ribonucleic acid (RNA). These are essential to all known forms of life since each plays an important and interdependent role in the cell. Proteins were first discovered in the early 1800s, and nucleic acids (DNA, RNA) were discovered in 1868. The important relationships between these macromolecules was not understood until many years later in 1956 when Francis Crick established the central dogma of molecular biology, which states the flow of genetic information in a biological system “DNA Makes RNA, and RNA Makes protein” [16] (Figure 1). Amazingly, this remains one of the central keystones of biology over half a century later. The steps in this process are now better understood as transcription, translation, replication and splicing. Years later in 1970 the field of bioinformatics was created to model and analyze life science data which was quickly growing too vast for human comprehension alone. A brief history of micro biology and bioinformatics follows.

**Figure 1**



*The central dogma of molecular biology demonstrates the relationships between DNA, RNA and protein [61].*

In the early 1800s proteins were discovered as a distinct class of biological molecules by Antoine Fourcroy [6]. Later, in 1838 the Swedish scientist Jöns Jacob Berzelius named this type of molecule “protein” based on the greek work “proteios” which means “primary”. In 1926 James B. Sumner demonstrated that the crystalline enzyme urease is also a protein [7], which was the turning point where the regulatory role of protein was first understood.

In 1868 Friedrich Miescher discovered nucleic acids, but the role of DNA and RNA in protein synthesis was not discovered until much later. In 1928, Frederick Griffith performed experiments where the traits of heated and

killed type III-S smooth, lethal *Pneumococcus* bacteria could be transferred to a type II-R rough (non-lethal) strain, which resulted in killing the host [8]. This led to a later discovery in 1943 where DNA (which was not destroyed by the heat) was discovered as to be the mechanism for transforming the previously non-lethal bacteria. In 1953 Francis Crick and James Watson presented the modern double-helix model of DNA in the *Journal of Nature*. Crick outlined the relationship between DNA, RNA and Protein and this became the basis of molecular biology in 1956.

In 1970 Paulien Hogeweg and Ben Hesper created the term “bioinformatics” to refer to the study of information in biotic systems. The modern meaning of the term is slightly different, and bioinformatics now combines various disciplines including computer science, engineering, statistics, and applied math to discover and process life science data. In bioinformatics, image and signal processing techniques are applied to biological data to simulate, model, and analyze Deoxyribonucleic acid (DNA), Ribonucleic acid (RNA), protein structures, and associated molecular interactions.

## Bioinformatics: Practical Applications

One goal of bioinformatics is to understand the ways in which various macromolecules of life interact, including protein, RNA and DNA. Within a eukaryotic cell there is a nucleus which contains chromosomes. These chromosomes contain most of the DNA within a living organism [5]. The DNA molecules consist of repeating patterns of cytosine (C), guanine (G), adenine (A), or thymine (T). Together, C, G, A, T encode the various instructions required to create and regulate life (Figure 2). A gene is a section of DNA which confers various traits of a species from parents to offspring. For example, the human genome contains 3.1 billion base pairs where about 2.9% encode genes and the other 97.1% (originally considered useless) contains instructions for when, where, and what volume of proteins to generate.

**Figure 2**

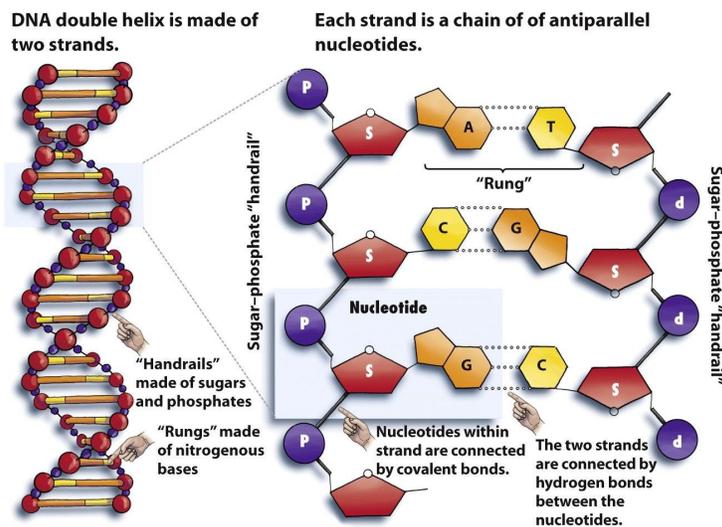


Figure 2-13ab *Biology: Science for Life, 2/e*  
© 2007 Pearson Prentice Hall, Inc.

*The double-helix structure of DNA [15].*

A protein is a large biomolecule which consists of long chains of amino acid sequences with a three-dimensional structure which is a result of the protein folds. The protein folds and resulting structure are determined by gene sequences, and thus, the genetic sequence determines both the form and function of proteins which are generated. Large proteins form important building blocks of life including muscle, cartilage, skin and hair. Smaller proteins such as hemoglobin, hormones, antibodies, antigens, and enzymes also play a critical role in regulating life.

As biological data continues to grow at an exponential rate, computers are becoming increasingly invaluable tools for analyzing various aspects of molecular biology, including genome sequences and macromolecular

structures. Bioinformatics is a broad field; however, the focus of this project is on predicting protein-compound interactions. For example, by designing a specific ligand which binds with a protein target, this can lead to the discovery of a new drug that changes the behavior of that protein and potentially cures a disease. There are a number of popular approaches for discovering protein-compound compatibility for drug discovery. The most common approaches include high-throughput screening (HTS) and virtual screening. Virtual screening includes docking methods, as well as machine learning based approaches.

**Figure 3**



*The Pivot Park ultra high throughput screening lab (uHTS) in the Netherlands [63].*

## **Bioinformatics: Experimental Approaches**

### **High Throughput Screening (HTS)**

By automating the process of running experiments with software, robotics, sensors, and liquid handling mechanisms, millions of chemical, genetic and pharmacological experiments can be performed quickly and accurately (Figure 3 illustrates an uHTS facility). Generally a clear plastic microtiter plate with 384, 1536 or 3456 wells is used. Each well is 9mm deep and contains the experimental matter, such as a chemical compound, protein, or cells (typically between 5 and 200 L per well). After sufficient incubation time has passed for the reaction to complete, measurements are taken from each well either manually or with a microplate reader. A microplate reader can use various detection modes to measure reactions, such as absorbance, luminescence, or fluorescence intensity [3]. An experimental compound with the desired reaction is considered a hit.

### **Artificial Enrichment: Docking**

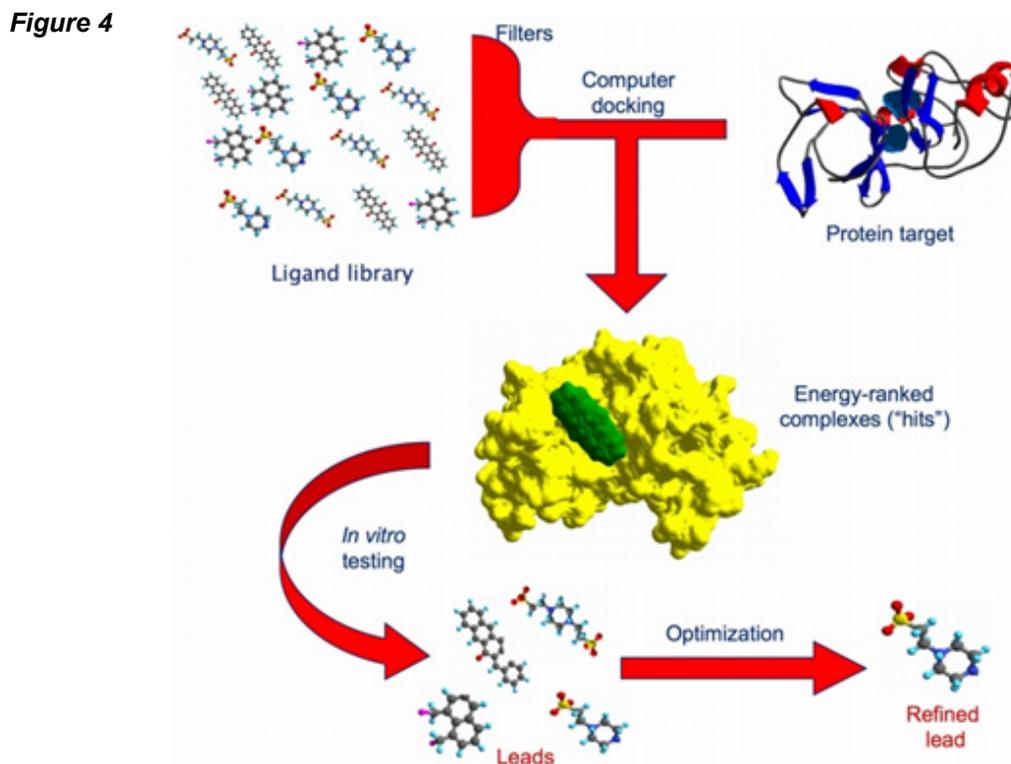
Brute force HTS is not financially viable for many institutions, and in the last decade an increasing number of researchers have turned towards computer-based enrichment methods to narrow down the number of compounds before running experiments. If a researcher can obtain the crystal structure of the target (e.g. protein fold structure), programs such as Autodock Vina can generate a three-dimensional model of the target and compare it the structures of various natural and designer compounds to determine whether physical docking is possible (Figure 4 illustrates the docking process). A computer simulation checks a series of docking poses to determine whether a given compound is structurally compatible with the target [4].

Typical docking software is unable to account for water displacement and various atomic level bonds which the three-dimensional structure simulation does not account for. In other words, although a compound might seem to be compatible with the crystal target structure, there may be some atomic level physics that block the binding. In practice, the UW-Madison SMSF is only able to achieve virtual enrichment in the 5 to 10 range with the docking approach.

### **Artificial Enrichment: Machine Learning**

Compared to most other scientific disciplines, artificial intelligence (AI) is still in its infancy. The field of AI did not formally exist until the mid-19<sup>th</sup> century when the first computers were created. Since that time both AI

and the sub-field of machine learning have rapidly evolved. As improvements in machine learning are discovered these advances are applied in many other fields including bioinformatics. Over the years, many different machine learning algorithms have been used to solve bioinformatic problems (such as artificial enrichment to predict protein-ligand binding). Examples include: fuzzy expert systems, genetic algorithms, inductive logic programming, logistic regression and decision trees.



*An overview of docking based virtual enrichment [13].*

Recent discoveries in deep learning show great potential for significant improvement over traditional virtual enrichment methods. To provide historical and theoretical context, we will introduce artificial intelligence, machine learning, as well as machine learning algorithms commonly used in bioinformatics. Next, we will lay the groundwork for deep learning by discussing multitask learning, perceptrons, and neural networks, as well as the key discoveries that made deep learning possible. Finally, we will discuss how multitask and deep learning algorithms were used in our experiments.

## Artificial Intelligence

What is artificial intelligence (AI)? According to John McCarthy (Figure 5), one of the first modern AI researchers, it is “the science and engineering of making intelligent machines, especially intelligent computer programs” [30]. The goal of AI is to create computers and software (intelligent agents), capable of learning and making intelligent decisions. An intelligent agent is an autonomous entity which observes its environment and makes decisions designed to maximize the possibility of success.

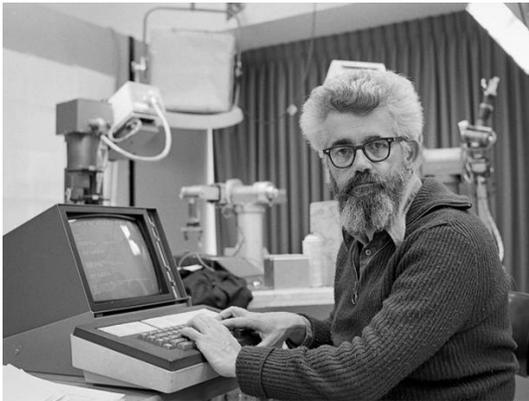
### Artificial Intelligence: History

The first general purpose computer “Electronic Numerical Integrator And Computer” (ENIAC) was created in 1941. ENIAC contained thousands of vacuum tubes, crystal diodes, relays, resistors and capacitors and occupied 1800 ft<sup>2</sup>. It was said that when ENIAC turned on, the lights in Philadelphia dimmed [20]. When

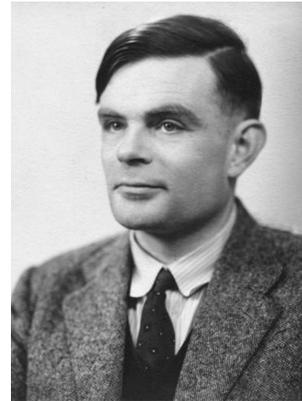
ENIAC was publicly announced in 1946, the media described the computer as a “Giant Brain” [20]. This designation is somewhat misleading considering this machine could not pass the Turing test for AI. The Turing test requires that both a human and an artificially intelligent machine carry on a conversation. If a human observer cannot decipher between the machine and the human, then the machine has passed the Turing test [21] (Figure 5).

In 1948 while working at the National Physical Laboratory in London, Alan Turing wrote a paper called “Intelligent Machinery, a Heretical Theory” in which he outlined various machine learning models which are variations on what are now called neural networks. The work was not published until 14 years after his death; some speculate this was because Sir Charles Darwin (grandson of the famous naturalist) dismissed his work as a “schoolboy essay” [23]. Alan Turing created and published his “Turing test” two years later in 1950.

**Figure 5**



*John McCarthy (1927-2011) is known as the father of artificial intelligence [31].*



*Alan Turing (1912-1954) created the “Turing test” to determine intelligence in 1950 [32].*

In 1956 the first research in artificial intelligence began with the Dartmouth Research conference. Tens of millions of dollars were awarded in research funding but the scientists over-promised stating an artificially intelligent machine as smart as a human being would be created within a generation [24]. The research failed to deliver satisfactory progress and research funding was withdrawn. In decades since then, financial sector support for AI research has been very cyclical. The period of 1974 to 1980 is described as the first AI winter since funding almost completely dried up, and the period of 1987 to 1993 is called the second AI winter. The term “AI winter” was coined by researchers who survived the long periods of funding cuts [24]. The intersection of statistics and computer science revived interest in machine learning during the 1990s, and the field shifted towards a more data-driven approach.

Although the failure to accomplish the grand promises of AI research sullied the reputation of the field in the past, industry and research success in more recent years has created a resurgence of interest in the field. Moore’s law and the evolution of computer processing speeds have also played a critical role in making modern AI a possibility. For example, in 1997 IBM’s Deep Blue computer defeated the current world chess champion Gary Kasparov, and in 2005 a Stanford robot won the DARPA grand challenge by driving itself through 131 miles of desert trails. In 2009 Google built a self-driving car, and in 2011 IBM’s Watson computer defeated the current Jeopardy game show champions. In 2011 smartphone applications included: Apple’s Siri, Google Now, and Microsoft’s Cortana, which all allow a human to interact with their smartphone device through natural language.

Since the late 1990s the rise of the internet and subsequent technology industry success with machine learning based applications has demonstrated that AI-based companies are economically viable. The rise of multibillion dollar web-based search companies such as Google, Yahoo, and Baidu demonstrate some significant progress milestones in AI such as web crawlers, data mining, and natural language web searches. Machine learning has

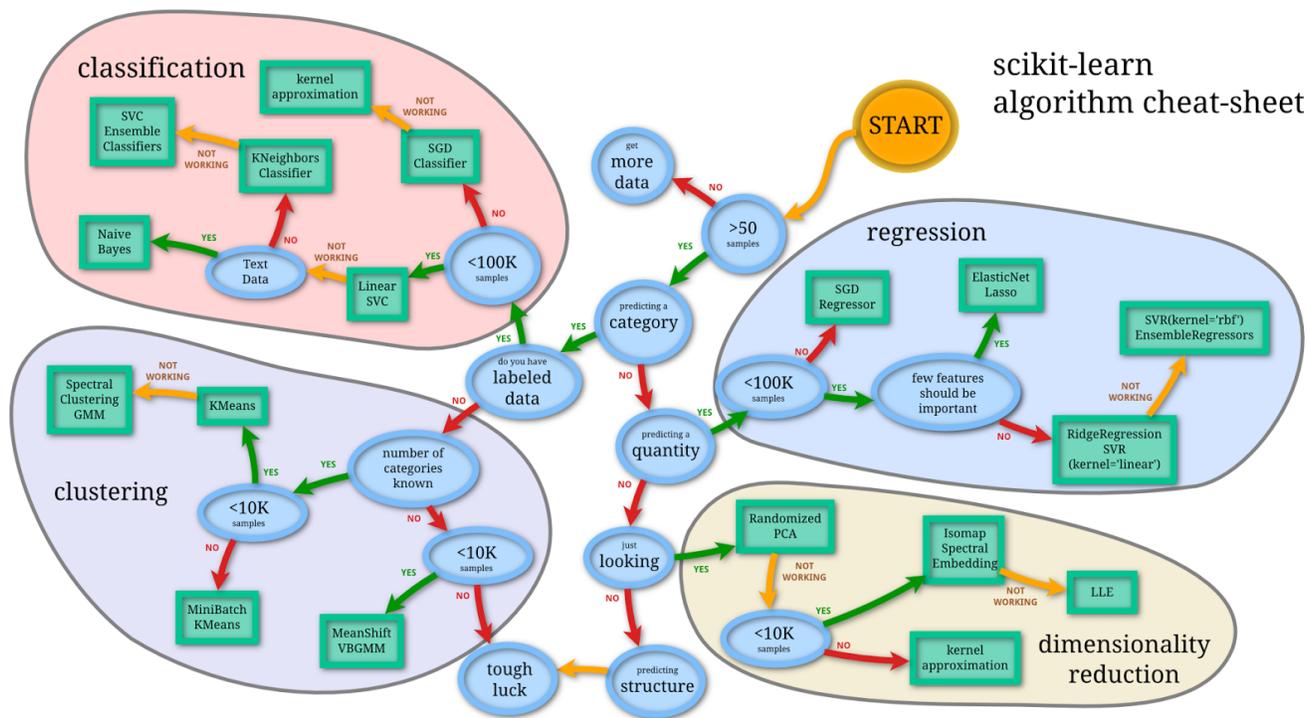
now become a working practical solution and many traditional organizations in both science and industry have started adopting machine learning approaches to solve data related problems.

As mentioned previously, many labs are experiencing cost challenges since the ratio of drugs discovered annually per billion spent have fallen in half each decade for several decades [2]. While big pharma has embraced brute force HTS solutions, smaller research labs have been forced to turn to more creative solutions. Recent advances in data-driven machine learning has the potential to help solve this problem by allowing these organizations to predict protein-ligand binding potential with computer simulations, narrowing down the pool of candidate choices for HTS resulting in a better cost-per-hit ratio. We will now take a brief diversion into the history of machine learning before discussing some of the machine learning methods commonly used in bioinformatics.

## Machine Learning: Introduction

Knowledge acquisition was identified as an artificial intelligence bottleneck early on in AI research; addressing this issue is the focus of machine learning. The bottleneck was caused by an exponential growth in data which resulted in similar growth in the need for expert systems. Machine learning originally evolved from statistical pattern recognition whereby a machine could use algorithms to recognize patterns in data. Machine learning extends the idea of pattern recognition to 'learn' from the past and predict future patterns.

Figure 6



This map, provided by Sci-kit learn, one of the most popular machine learning frameworks, helps a practitioner determine which machine learning algorithm is most suitable given a problem domain [37].

The goal of machine learning (ML) is for machines to learn from experience and act on that learning without being explicitly programmed for that. In other words, the goal is to create machines that learn implicitly from algorithms with little or no human supervision. This is in contrast to expert systems which must be hand-crafted based on the knowledge of experts (Figure 6 shows heuristics for choosing the right ML algorithm).

In more formal terms, machine learning is the study of algorithms that improve their performance  $P$  at some task

$T$  with experience  $E$ . A well defined machine learning problem clearly specifies  $\langle P, T, E \rangle$ . For example, the task might be predicting how much a consumer might enjoy reading a book ( $T$ ), given a user-history of past book ratings ( $E$ ), and the difference between the predicted and actual rating is performance ( $P$ ). There are many different kinds of machine learning algorithms and some are more effective in some contexts than others. Some examples of common machine learning algorithms are: inductive logic programming, logistic regression, decision trees, random forests and multitask learning.

There are two primary types of machine learning — supervised and unsupervised. In supervised learning,  $Y$  is predicted given a feature vector  $X$ . For example, given the features of a molecular compound such as atomic structure and chirality, one could try and predict if the compound will bind (active = 1) or not bind (inactive = 0) with a target protein  $Y$ ; this is a binary classification task  $\{0, 1\}$ . One could then use feature vectors from many known compounds that were already tested in a lab (and determined as active or inactive) to train a machine learning model. When  $Y$  is already known for a particular dataset, it is referred to as “labeled” data. After the model was trained with the labeled data one could then input a new untested molecule into the model and predict whether or not the compound will be active or inactive (will it bind with the protein target). If the machine learning model tends to make accurate predictions about binding potential (active / inactive), then the model generalizes well.

In contrast, with unsupervised learning, the data has no labels. In other words, with the compound-protein binding example, the algorithm would only be given the molecule vectors  $X$  without any classification preference  $Y$  for the protein. A machine learning algorithm such as  $k$ -means clustering could then be applied to find interesting patterns or groupings within the unlabeled data. Unsupervised learning has many bioinformatics applications, especially for experiments with vast amounts of unlabeled data. For example, recent work by Michael Newton et al. demonstrated that unsupervised compound clustering can be used to improve docking score enrichment [64].

Aside from supervised and unsupervised learning, there are also semi-supervised hybrid machine learning algorithms that use a small amount of labeled data and a large amount of unlabeled data. Since this project uses a supervised machine learning approach (vast quantities of labeled data are available), we now shift our focus to a number of common supervised machine learning approaches used in bioinformatics.

## Artificial Enrichment: Machine Learning

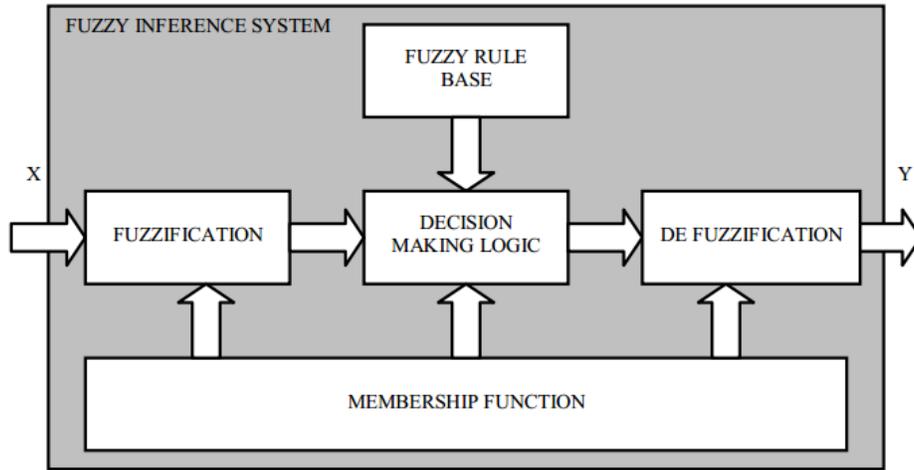
As mentioned above, artificial enrichment *in silico* methods can predict potential protein-compound binding in a computer simulation. Aside from HTS and docking approaches, machine learning is another common virtual enrichment approach. Examples of machine learning algorithms used in bioinformatics include fuzzy expert systems, genetic algorithms, inductive logic programming, multitask learning, random forests, logistic regression, and artificial neural networks (ANNs).

## Fuzzy Expert Systems

In 1965 Lotfi A. Zadeh, a University of California at Berkeley professor, proposed fuzzy sets as an extension to classical set theory in mathematics [10]. A fuzzy set contains entities with a continuum in grades of membership as opposed to traditional boolean thinking where something is either a member of a set or it is not. Zadeh designates traditional sets with absolute membership in binary  $\{0, 1\}$  terms as “crisp sets”. Concepts such as union, intersection, convexity and relations were extended to fuzzy sets and the term “fuzzy logic” was introduced [10]. This forms the basis for fuzzy expert systems which can deal with “fuzzy” (i.e. partially true) concepts. Expert systems use explicit knowledge as opposed to most other algorithms which tend to use implicit knowledge (Figure 7 illustrates a fuzzy expert system).

One advantage of fuzzy systems is that decisions with degrees of uncertainty can be implemented in ways humans can understand, which allows the experience of human experts to be incorporated into these systems. Fuzzy systems can also deal with noisy data and uncertainty when dealing with a variety of biological patterns.

**Figure 7**



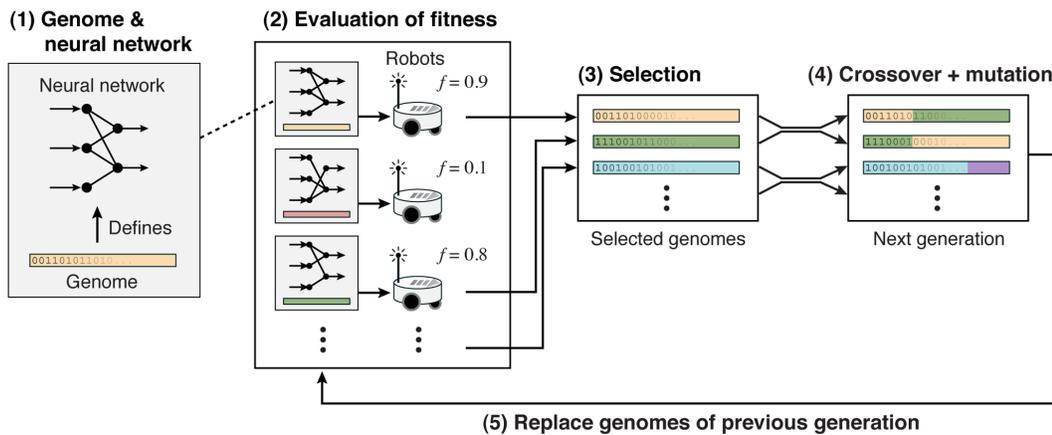
*The block structure of a fuzzy logic system [12].*

One example application was a fuzzy expert system built in 2000 by Woolf, Peter J., and Yixin Wang. The authors used a fuzzy logic approach to analyze yeast gene expression data, which could sometimes predict the functions of unknown genes [11]. Fuzzy logic has also been used to enhance docking scores for virtual enrichment. For example, In 2009 a study by David Hecht et al. contrasted fuzzy logic, artificial neural networks and evolutionary computation for calculating improved docking scores [65].

### Genetic Algorithms

Genetic algorithms are heuristic methods that simulate natural selection within a reproducing population. This is done by generating a population of individuals encoded as strings with various possible biological traits. A fitness function determines how likely these individuals are to reproduce and provides a measure of goodness which represents a global optimum to work towards. The least fit individual drops out of the population and is no longer able to reproduce. The remaining individuals go through a crossover state where individuals reproduce and mix their genetic traits which are inherited by their offspring. A mutation phase follows where with a very small probability that some traits may change. If any individuals have achieved the predetermined optimal level of fitness (e.g. some set of “good” traits), then the algorithm stops. Otherwise, the algorithm keeps running through additional generations and keeps reproducing offspring and dropping unfit individuals until an optimally fit individual is produced.

**Figure 8**



*The standard genetic algorithm in the context of Darwinian selection applied to robots [19].*

Genetic algorithms are used in bioinformatics for multiple sequence alignment (MSA), gene prediction, and

population genetics modeling. By using MSA, shared evolutionary origins can be inferred from DNA, RNA, and protein sequences. The goal of gene prediction is to discover what regions of genomic DNA encode genes. Population genetics modeling is a highly mathematical discipline where allele frequencies in a population are studied. Through reproduction each parent contributes one allele to an offspring and the net effect of allow reproduction in a population directly impacts the allele distribution within a population. Genetics modeling also studies the various factors that affect allele frequency such as natural selection, sexual selection, mutation, genetic drift and gene flow.

In 2010 two Swiss researchers applied a genetic algorithm to a population of robots with various traits including navigation, homing, predation, brain and body morphology. They found that after a few hundred generations, the robots were able to adapt strategies for hunting and evasion, and were able to navigate a maze without bumping into any walls. The bots even manifested traits such as cooperation and altruism. The graphic (Figure 8) from the study provides a good visualization of how genetic algorithms work, although their specific application is unusual (robot evolution).

Genetic algorithms are also commonly used in docking approaches (virtual enrichment) to predict protein-ligand binding. For example, a 1998 study by Morris et al. compared Monte Carlo simulated annealing, traditional genetic algorithms and Lamarckian genetic algorithms to predict the “bound conformations of flexible ligands to macromolecular targets” [66].

## Inductive Logic Programming

In 1991 Stephen Muggleton coined the term inductive logic programming (ILP), defined as the intersection between logic programming and machine learning [34]. Inductive logic programming has proven effective for many bioinformatics and natural language processing tasks. ILP works by providing many positive and negative examples to an inductive learner which builds logical rules that fit the data. In other words: *positive examples + negative examples + background predicates => hypothesis*. Variations of inductive logic programming include: inverse resolution, GOLEM, FOIL, CHAM and CHILLIN.

Inverse resolution and GOLEM are considered bottom-up ILP systems because they start from the most specific clauses that correctly identify positive training examples and generalize until any additional generalization would result in misclassification of negative examples. FOIL and CHAM are examples of top-down ILP systems. In a top-down ILP system “a specialization operator *S* produces a set of clauses *C* which are allowable by the language bias from a clause *c*” [36]. CHILLIN implements a hybrid learning model with combined aspects of both top-down and bottom up ILP logic [35].

The First Order Inductive Learner (FOIL) is one of the most popular types of ILP and works by using both positive and negative training examples for a target concept, as well as background knowledge predicates to learn clauses that only identify positive tuples. FOIL is known as a “function free” ILP method because it cannot use any constants or function symbols [35]. Foil applies a separate-and-conquer strategy (as opposed to the more typical divide and conquer) since each iteration of the algorithm adds one rule at a time until there are no positive examples (or few) left.

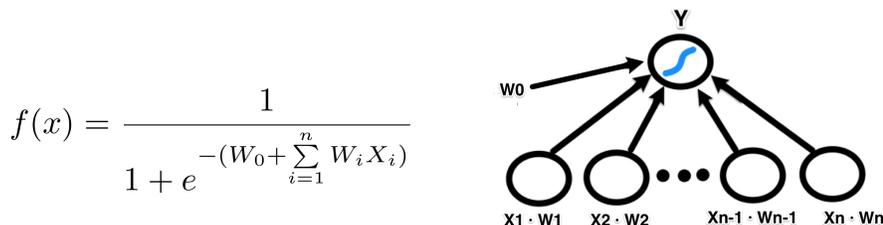
In 2010 David Page et al. created an ILP-based model of hexose-binding sites and compared the results to several baseline “black box” machine learning methods [67]. The ILP-based method performance was similar to the baselines, but ultimately the ILP-based results were more useful since it gave insight into the way the model was making decisions. Although ILP is an effective algorithm for solving bioinformatics problems it is not a common method. Thus, ILP was not chosen as one of our baseline methods.

## Logistic Regression

The name logistic regression is a misnomer because it is used for classification (not regression). Logistic

regression is for datasets with 1 or more independent variables (the feature vector  $X$ ) and a binary dependent variable  $Y$  with two mutually exclusive outcomes; for example: true or false, active or inactive. Binary logistic regression is analogous to linear regression except that the dependent variable  $Y$  is a measurement (instead of a nominal 0 or 1). Logistic regression uses the logistic function since all numbers using this function up to infinity map between 0 and 1 respectively, based on a smooth s-curve. This is ideal for keeping all resulting values within probabilistic bounds.

**Figure 9**



Left: the sigmoid logistic function. Right: feature vectors from  $X$  are multiplied with the respective weight vectors  $W_i$  before being passed to the logistic activation function. Then a threshold is applied to determine the class.

The logistic regression function  $f(x)$  calculation is shown in Figure 9 on the left-side. The right-side of figure 9 is a visualization of the input feature vector  $X_1 \dots X_n$  and the logistic function which is applied to the input vector and corresponding weights to produce an output between 0 and 1 resulting with a  $Y$  classification based on the threshold function (e.g. 0.5). The bias unit  $W_0$  is used to shift the entire curve left or right, and the remaining weights  $W_i$  then help determine how probable the target  $Y$  is based on the features  $X$ .

Logistic regression is relatively easy to implement (compared to other machine learning algorithms) and can usually generate reasonably accurate classification results for many different types of bioinformatics problems. These were some of the reasons for our decision to include logistic regression as one of our baseline methods for this project.

## Decision Trees

The goal of decision trees is to apply a divide-and-conquer approach and make the tree as small as possible, while still correctly classifying the training set; unfortunately this is an NP-hard problem. ID3, C4.5 and CART are three of the most common decision tree types. Instead, ID3 decision trees use a heuristic: greedily choose splits that maximize information gain. Information gain is maximized by reducing the entropy (uncertainty) of random variables in the dataset.

In information theory, each possible variation of a feature is encoded with a sequence of  $\{0, 1\}$  bits. Instead of giving all features an equal number of bits, the least bits are given to the most common features and progressively more bits to those features that are the least common.

Entropy can be measured as the expected number of bits to encode the variable. Conditional entropy can be determined if the additional step of conditioning on some other variable is added. Choosing the splits that reduce conditional entropy the most is known as mutual information gain [41] (Figure 10), and this is how splits are chosen in ID3 decision trees. In other words, at every step the loop considers each possible split and chooses the split that maximizes information gain.

The weakness of information gain is that some features are unique for every instance; gain ratio is used to

**Figure 10**

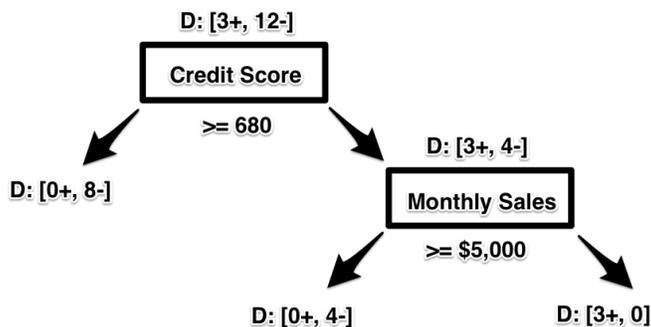
$$\text{Entropy } H(Y) = - \sum_{y \in Y} P(y) \log_2 P(y) \quad \text{Conditional Entropy } H(Y|X) = - \sum_{x \in X} P(X = x) H(Y|X = x)$$

Above: the entropy and conditional entropy equations used to calculate mutual information for decision tree splits.

overcome this limitation. Finally, the algorithm stops splitting when all the remaining examples only fit one class.

The example decision tree below determines whether a business might qualify for a \$15,000 loan based on credit score and monthly sales. The training set  $D$  contains 3 positive and 12 negative examples. Notice how the credit score feature filters a larger number of applicants than monthly sales; hence, the tree splits on this property first to maximize information gain. Although there might be many more features, there is no need to split beyond monthly sales since all examples are now correctly classified.

**Figure 11**



Above: an example of a decision tree to determine if a borrower is qualified to obtain a \$15,000 loan.

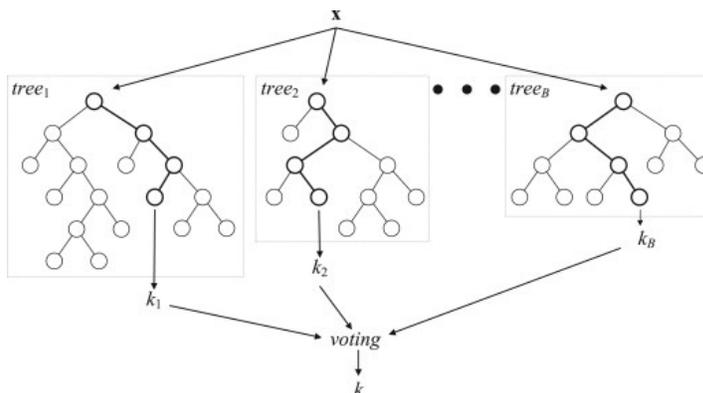
One of the greatest strengths of decision trees is that the algorithm makes decisions that are inherently easy for humans to understand (see Figure 11). Also, decision trees are often a good choice when data is sparse, but as the size of the training set increases, decision trees tend to overfit data. Decision trees greedily attempt to maximize information gain and this creates an inherent inductive bias in the way decisions are made. This means the types of hypotheses a decision tree favors will tend to overfit training examples as the size of the dataset increases.

### Random Forests

Decision trees (especially deep ones) tend to have low bias and high variance; random forests are an ensemble method that overcomes these limitations. This reduces the variance in the overall model at the cost of slightly increased bias and somewhat less comprehensible decisions (since many decision trees are involved in the final prediction). There are two typical ways random forests randomize the results: bootstrap aggregation and random feature selection.

Bootstrap aggregating (or bagging) builds decision trees on subsets of the data which results in many different

**Figure 12**



Above: random forest visualization where many decision trees are built from the training data vectors  $X$  and result in a majority vote prediction for target  $K$  [39].

random decision trees which helps mitigate bias (Figure 12 illustrates a random forest). With bagging a random sample is drawn from the training data, with replacement, to train a number of different decision trees (typically 10, 20, 50 or 100). In addition to bagging, random forests also select features from a random subset of features to help get more variation in the kinds of decisions that are made. Sometimes this process is referred to as feature bagging because this is just like training set bagging, except it is applied to features. Predictions can then be made with the combined results of all the decision trees with the mean, or a majority vote. Other common methods of randomizing forests include random split selection and random training weights [40].

Random forests do not produce stellar results, but they are robust in the sense that they can be applied to almost any problem domain and produce “pretty good” (but not great) accuracy. Generally, a domain specific machine learning algorithm should (almost) always outperform random forests. This along with ease of implementation are some of the reasons we choose random forests as one of the baseline methods for this project.

## Multitask Learning

In traditional single task learning, one trains examples for a single target and a prediction is made based on that one target. In contrast, with multitask learning, a mixture of training examples are drawn from many related problems (that share the same set of features), and predictions are made for many different tasks at once. Assuming the various tasks share some latent shared model, the learner can use the training examples to improve generalization.

The multitask method of learning many similar problems at once to become more effective at solving related problems is a type of inductive transfer. Multitask learning can be combined with any machine learning method that can generate multiple outputs. One way to translate multiple outputs into probabilities is by using a softmax function. A softmax generates probabilities for mutually exclusive numeric outputs by normalizing over the total output (for each respective outcome). For example, a bioinformatics multitask learning model could predict several mutually exclusive outcomes such as the probability a compound will bind with several respective targets, all of which are in the same host (where the compound could only bind with one protein).

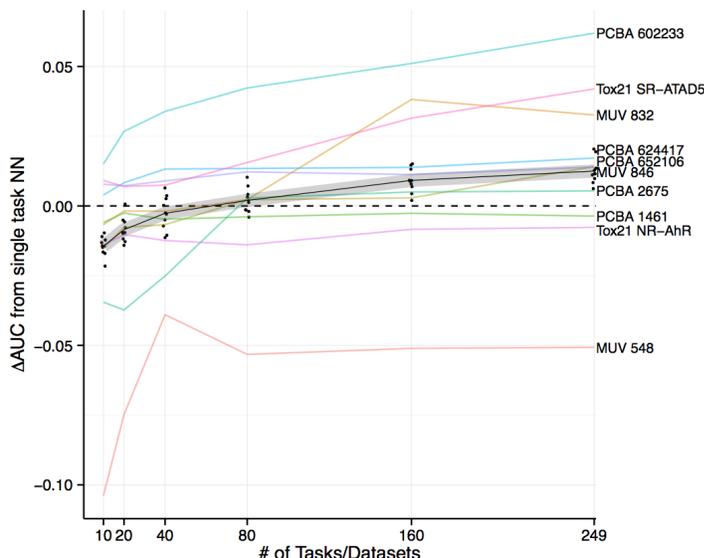
Multitask learning has been applied to many different bioinformatics related problems including genome-wide association studies, protein structure, protein-protein interactions [68]. In 2014 Anthony Gitter et al. developed MT-SDREM, a multitask learning model for deciphering signaling and regulatory networks. Their experiments included the application of SDREM to decipher human auto-immune responses to three flu-stains: H1N1, H5N1 and H3N2 [68].

Several years ago there was an extreme sparsity of multitask research in bioinformatics, but this is starting to change. For example, in 2014 Geoffrey Hinton’s group won the Merck Kaggle challenge by applying a multitask learning strategy to deep network models [48]. The Merck Kaggle challenge involved predicting compound-target binding for 15 different assays. Training the QSAR (Quantitative Structure-Activity/Property Relationship) model on all 15 tasks at once regularized the model (i.e. prevented overfitting) and forced the system to generalize more effectively and learn the latent structures in the data.

Recent research such as the Stanford-Google MMNT experiments discovered that adding large numbers of additional tasks to similar bioinformatics problems combined with deep learning (a type of neural network explained below) can scale extremely well. The MMNT experiments used data for 259 targets (mostly proteins) and 1.6 million compounds to predict whether unseen compounds would likely bind with the target (active or inactive). Each additional target dataset continued to increase accuracy measured as area under curve (AUC) as the number of targets increased [1] (Figure 13 displays the MMNT AUC results).

AUC is an accuracy metric commonly used in binary classification problems. AUC is obtained by taking the integral of an ROC (the receiver operating characteristic) curve. An ROC curve plots the false positive rate (FPR,

**Figure 13**



Above: multitask results from the Google-Stanford MMNT experiments. The Y-axis is AUC, and the X-axis is the number of tasks. The AUC accuracy continues to increase as additional tasks are added and the plateau is not in sight [1].

x-axis) against the true positive rate (TPR, y-axis). The ROC curve shows how a binary classifier performs as the discrimination threshold varies. The intuition for AUC is this: given two random examples, one of which is positive and the other is negative, what is the proportion of the time one guesses correctly?

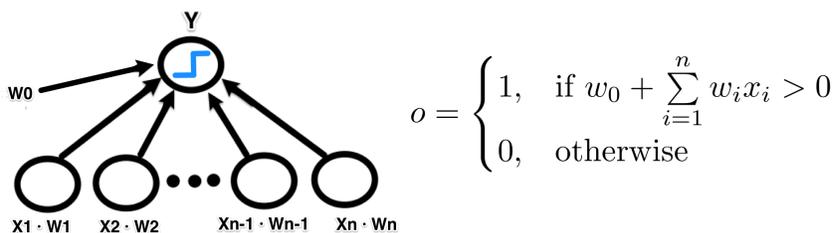
One benefit of AUC is that it cannot be easily manipulated like raw accuracy. For example, in a scenario where 99% of all examples are positive, a classifier could always guess positive no matter what and still achieve 99% accuracy. AUC provides a more robust measure of accuracy by preventing this kind of manipulation (in this specific example the guess is always positive so the AUC will be 0.50). The high AUC scores achieved (as measured verse baseline methods) by the Stanford-Google MMNT team demonstrated that multitask learning works well when combined with deep learning.

Multitask learning is one of the algorithms used in this project due to the strong synergy that can be achieved by learning many tasks at once, combined with a deep learning model which is uniquely capable of utilizing the data across many tasks. Before we discuss deep learning further, we will begin with the fundamental building blocks of neural networks (the perceptron) as well as some of the history and theory to provide context.

### Neural Networks: The Perceptron

Artificial Neural Networks (ANNs) are inspired by interconnected neurons in biological systems which receive some number of real-valued inputs and translate that into an output. In a very generic sense, this mimics the way neurons work in biological systems. In 1943 McCulloch and Pitts [47] created the original mathematical model for neural networks based on binary threshold logic (as shown in Figure 14). In 1958 Frank Rosenblatt

**Figure 14**



Left: a visualization of perceptron math. Right: the binary threshold identity function.

published his work on one of the first machine learning algorithms, the perceptron [25].

Many grandiose predictions were made regarding the predictive power of perceptrons, and in 1969, Marvin Minsky and Seymour A. Papert wrote a rebuttal which demonstrated many of the limitations of perceptrons [28]. Any concept that is not linearly separable cannot be learned by a single perceptron; the XOR example is a common example of a non-linearly separable problem. The XOR problem is unsolvable for a single perceptron. Minsky and Papert's criticisms became generally accepted at the time, and perceptron-based models fell into disrepute for years. This led to a major decline in neural network related research for decades.

In 1986 backpropagation was discovered by Rumelhart, Hinton, and Williams, which allowed neural networks to train hidden layers [29]. This debunked the previous limitations of perceptron-based systems as outlined by Minsky and Papert. Figure 14 (on the left-side) shows the math in a perceptron which is somewhat similar to logistic regression, except the activation function is a binary threshold [47]. The identity function of the binary threshold is shown in Figure 14 (on the right-side); the result can only be  $\{0, 1\}$ . During training the output  $o$  is compared to the expected target value  $y$ .

The perceptron learning update rule is as follows:

1. *Randomly initialize weights.*
2. *Calculate the output for the given instance using the formula in Figure 14.*
3. *Update the weights:*

Given the learning rate  $\eta$ :  $0 < \eta < 1$

Determine the change in weight:  $\Delta w_i = \eta(y - o)x_i$

update the weight:  $w_i \leftarrow w_i + \Delta w_i$

Perceptrons are used as building blocks for much larger, more complicated neural networks. Many different kinds of models can be built with perceptrons; the behavior can be modified depending on the type of activation function used. Some common activations include: hyperbolic tangents, ramp functions, step functions, Gaussian kernels, and rectified linear units (ReLU). A rectified linear unit (ReLU) creates a weighted sum of the linear units and returns the total if it is greater than zero; otherwise the result is 0. Some argue that ReLU are more biologically plausible than the more widely used sigmoid activation.

## Artificial Neural Networks

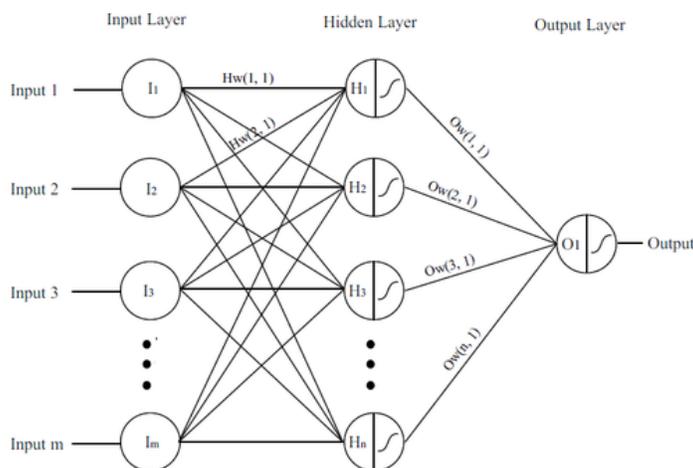
The power of neural networks lies in their ability to make multiple non-linear transformations through many layers of neurons which can represent complex and increasingly abstract features (as more layers are added and as layers are made wider by increasing the number of neurons). By adding many hidden layers of varying widths to a neural network, the model can learn increasingly complex and abstract representations. However, in order to use hidden layers one must find a way to determine how to assign error attribution and make corrections working backwards through the neural network. This backwards propagation of errors is known formally as "backpropagation" [29].

Although the conceptual foundation of backpropagation was discovered in 1963, it was not until 1986 that Hinton et. al discovered a way for this algorithm to be applied to neural networks [29]. Backpropagation made it possible to overcome the representational limitations of perceptrons by adding additional hidden input layers (Figure 15 displays a sigmoid neural network with a single hidden layer). During the process of backpropagation, gradient descent is used to find the minimums in the error surface caused by each respective neuron when generating a corresponding output. In order to use gradient descent and minimize errors, the activation function must be differentiable, which is one reason why sigmoid activation functions are a common choice.

Gradient descent is an iterative process that is used to find where the error surface descends most steeply.

Gradient descent then takes an iterative step in that direction by updating the weights. This is done by progressively working backwards through the model (figures 16 and 17) through subsequent hidden layers, moving towards the original inputs. One problem with gradient descent is that the error signals become progressively more diffused as the signal goes back through each progressive hidden layer. This is because, as the signal goes deeper into the model, an increasing number of neurons and weights are associated with a given error. This makes it difficult to train more than two hidden layers efficiently. Although hidden layers three levels and beyond can be trained, by the time a good convergence is reached in the deep layers, the layers closest to the output are likely to be overtrained.

**Figure 15**



*A single layer neural network using sigmoid activation functions. Input units are on the far left and sigmoid activation functions perform non-linear transformations on the inputs [43].*

## Deep Learning

In the last decade deep learning has become the dominant method for speech and audio recognition. In more recent years deep learning has also shown promise in many other disciplines. Some examples include: the analysis of particle accelerator data, predicting the activity of potential drug molecules, reconstructing brain circuits, and predicting user movie preferences (e.g. Netflix [69]). Deep learning is poised to leverage a combination of multitask learning, automatic feature extraction, and progressively more abstract concepts (e.g. feature hierarchy) which make it an ideal choice for many learning tasks beyond audio and visual recognition.

Deep learning is part of a family of machine learning algorithms that transform inputs and reconstruct them into a distributed representation. In a distributed representation, many concepts are associated with many neurons and many neurons are associated with many concepts. By using good random initialization, symmetry is intentionally broken so that many different features will be learned by the neurons in the various layers of the system; this is referred to as automatic feature detection. Each hidden layer becomes the input into the next layer of the system and non-linear transformations can be applied which allows a system to progressively learn increasingly abstract and deep features (which is why it is called “deep learning”) (Figure 16 illustrates how a deep neural network learns increasingly complex and abstract features by adding additional layers).

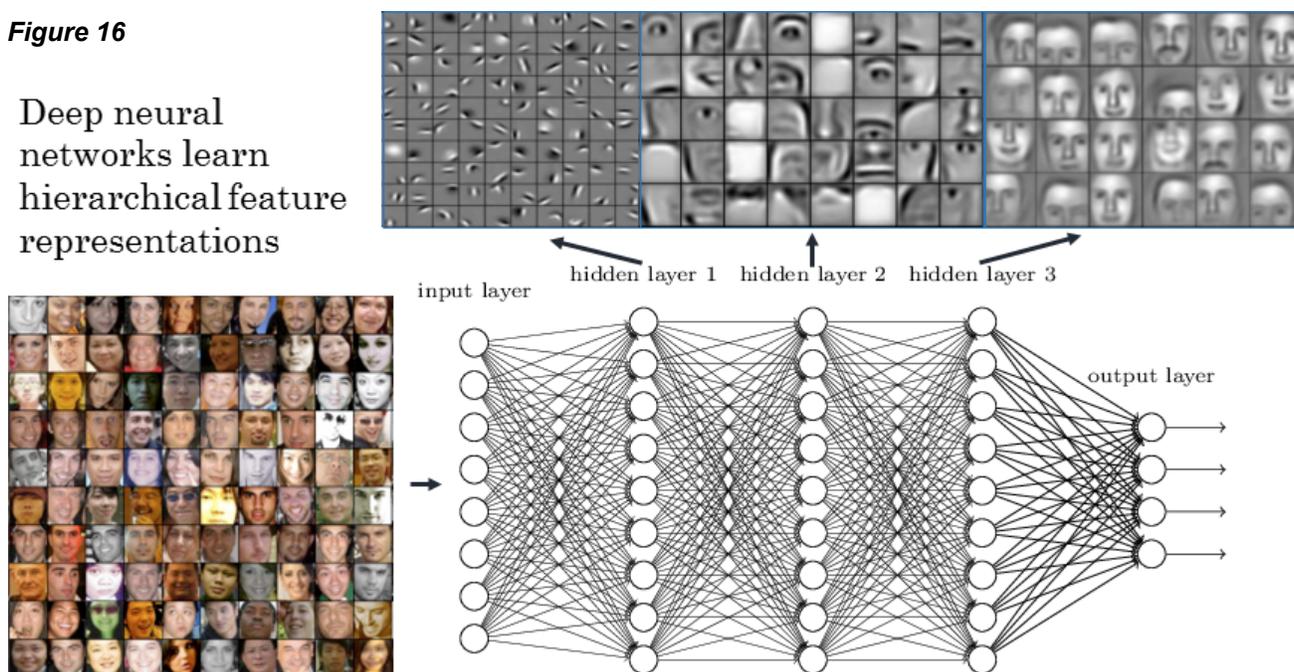
Automatic feature detection is one of the most significant benefits of a deep learning system. A traditional machine learning model must be carefully constructed with the right features in order to generalize well (a process referred to as feature extraction or feature engineering). This can be extremely time consuming and requires domain expertise, which is not practical long term as data in many fields continues to grow at an exponential pace. For example, in 2014 Hinton’s group won the Merck Kaggle challenge without any feature engineering (this is also mentioned in a different context in the multitask learning section above) [48].

Traditional machine learning models such as decision trees, SVMs, and  $k$ -means or  $k$ -nearest neighbor make intuitive sense and generally create models that humans can easily understand. One problem with deep learning is automatic feature detection can discover features that are difficult to decipher. Another issue with deep learning is complex, deep and wide neural networks with many neurons often have extreme memory and processing power requirements. This can potentially create scalability problems in systems where millisecond response time is critical for a good user experience.

Some of the speed issues with deep learning can be mitigated by using CUDA capable software with high power Nvidia graphics cards which can run millions of matrix computations in parallel with extremely high speed. Most of the mainstream deep learning frameworks support this. Some of the leading deep learning frameworks include: Torch (Facebook), Theano (Open Source Community), and DL4J (Skyrmind). Although deep learning is just one type of machine learning, it is so completely different in its computational requirements and setup that it requires different frameworks entirely. This may change in a few years, but as of this writing, it seems none of the mainstream machine learning frameworks such as Weka or scikit-learn support deep learning.

**Figure 16**

Deep neural networks learn hierarchical feature representations



*An example of how a deep learning model can learn progressively deeper hierarchical features. The lowest layers learn very simplistic features and provide inputs to the higher layers which learn more abstract features. In this image, the first hidden layer only detects edges or other low level details. The second layer detects more abstract features such as eyes, mouths and noses. In the final layer, abstract representations of various faces are reconstructed for facial recognition [51].*

## Deep Learning: Optimization

There are a number of ways to accelerate the deep learning process, reduce overfitting, and converge on more optimal results. Unsupervised pre-training, weight decay and dropout are common regularization methods that help prevent overfitting. During dropout a fixed proportion (e.g. 0.25) of neurons are randomly selected to be temporarily excluded from the model. This has the effect of simulating many different (heuristically infinite) architectures during training; this prevents co-dependency among neurons and regularizes the model (prevents overfitting) [70]. Unsupervised pre-training (explained below) provides several benefits including: better weight initialization, more local optima, and faster training. Nesterov's accelerated gradient descent or other classical momentum methods can also help gradient descent methods reach convergence in less training epochs. Finally, annealing the learning rate through step decay or exponential decay can help the model settle

into a deeper, narrower part of the loss function.

Pre-training, discovered by Hinton et al. in 2006 is a fast, greedy, algorithm that uses an unsupervised layer-wise approach to train a deep neural network one layer at a time [44]. In the case of a DBN, each layer is trained with contrastive divergence in an unsupervised manner and each layer becomes the input for the next layer during the process. The process for a deep autoencoder is the same except each layer is trained with with a standard sparse autoencoder algorithm (autoencoders are discussed in the next section). After the pre-training phase is complete, a slower fine-tuning process, such as stochastic gradient descent (SGD), is used to train the model. With the pre-training approach, the model has already learned the features before backpropagation begins. In this setting, backpropagation is only used on the two outer-most layers to fine-tune the parts of the model which are relevant to the current classification task.

Although unsupervised pre-training is considered “obsolete” [71], it is still required for any deep learning architecture that uses activation functions that suffer from gradient-based problems (such as sigmoids). For example, the sigmoid non-linear smoothing function has an increasingly flat gradient as numbers grow towards positive or negative infinity, and numbers towards the middle of the sigmoid (0.5) have steep gradients. During the SGD backpropagation process, sigmoid activations are not a problem for a one or two layer model. However, once the signal propagates through three layers or more, it tends to explode or vanish, distorting the data and resulting in a model that fails to generalize well. Thus, with three or more layers of SGD backpropagation the model can get caught in plateaus and local minima. Greedy unsupervised layer-wise pre-training bypasses this problem by training each layer directly.

ReLU-based neural networks do not require pre-training since this type of activation is not subject to the vanishing / exploding gradients problem. Typically, dropout and early stopping are enough to keep a ReLU-based architecture regularized [71, 72]. Regardless of whether pre-training is applied, good random initialization remains a critical step in the process because it breaks symmetry in the model increasing the odds that no two neurons will learn the exact same thing, which is important for automatic, distributed feature-detection. If a neural network is initialized poorly, data might not flow through the network to the final layers and the learning model is more likely to fall into local minima and generalize poorly.

## Neural Networks: Deep Belief Networks

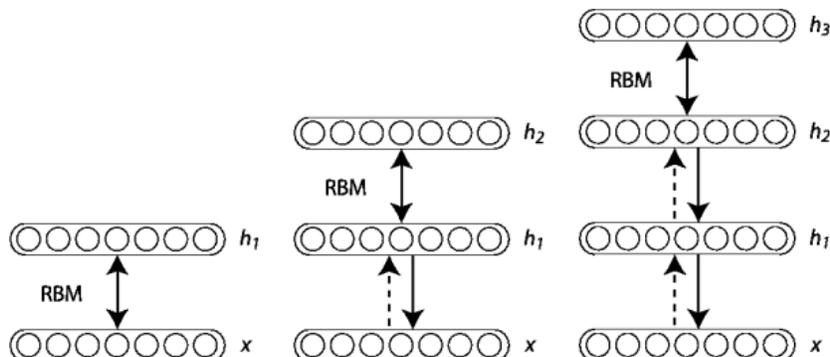
A Boltzmann machine is an energy-based model proposed by Hinton et al. in 1985 [53]. A Boltzmann machine works based on the concept of thermal equilibrium; everything is defined in terms of the joint configurations between visible and hidden units. In a Boltzmann machine, all visible units are connected to all hidden units, but the hidden units are also connected to each other. Updates are performed through respective positive and negative phases. The positive phase finds configurations that work well and updates their energies, while the negative phase unlearns spurious minima. During the positive phase, a set of particles is saved for each training case and each particle contains the current value of a hidden unit. During the negative phase, a set of fantasy particles are stored where each has a global configuration. The simple mean field approximation is a heuristic that allows updates with probabilities instead of binary states. The fundamental problem with Boltzmann machines is the update algorithm is very slow, and they were rarely used in practice.

In 1998 Hinton et al. discovered the restricted Boltzmann machine (RBM) which is almost the same design as the Boltzmann machine, except there are no connections between hidden units. The restricted Boltzmann machine is a bipartite graph where each visible unit is connected to each hidden unit with undirected connections. Hinton discovered that by clamping the data vector, thermal equilibrium could be reached in a single step through a process he calls contrastive divergence. Although we cannot evaluate the energy function directly, contrastive divergence can estimate the gradient of the energy function regardless.

The contrastive divergence shortcut has theoretical problems, and although it works like maximum likelihood approximation in practice, it does not actually compute the log likelihood of the gradient. The key benefit of this

shortcut is it allows all the updates to be performed in parallel which makes the process vastly more efficient than the original Boltzmann machine (i.e. far less computational complexity). This was an important discovery because it made the RBM algorithm computationally efficient enough to be used in a practical setting. In fact, RBMs are becoming widespread in practice and in 2009 RBMs were part of the ensemble that won the 1 million USD bounty in the Netflix Challenge.

**Figure 17**



*A deep belief network (DBN) has an input vector  $x$  with RBM layers  $h_1$ ,  $h_2$ ,  $h_3$  stacked one on top of the other. Each layer becomes the input for the next layer [45].*

In 2006 Hinton et al. discovered a way to stack RBMs and train deep belief networks (DBNs) [44]. A DBN is probabilistic generative model with multiple layers of stochastic latent variables. Typically, the latent variables have binary values and are referred to as feature detectors (or hidden units). A DBN is constructed with an input layer followed by a number of stacked hidden layers, each of which is an RBM (displayed in Figure 17). Each subsequent layer becomes the input for the next layer. The weights are randomly initialized to break symmetry and help the hidden units learn many different features. Greedy layer-wise unsupervised pre-training with contrastive divergence is used to help the DBN reconstruct the inputs for feature detection.

Figure 17 [45] illustrates the unsupervised pre-training process which begins with the input layer which has some form of training data (e.g. pixels from an image). The weights of the first RBM (hidden layer  $h_1$ ) are then trained with contrastive divergence to reach a state of energy equilibrium based on the input data. Then the hidden activity pattern (the resulting binary states) of  $h_1$  becomes the input for the next hidden layer  $h_2$ , and contrastive divergence is applied again. This process continues until the final hidden layer is trained (which is  $h_3$  in this example). This results in setting the weights to more suitable starting positions before the fine-tuning process starts.

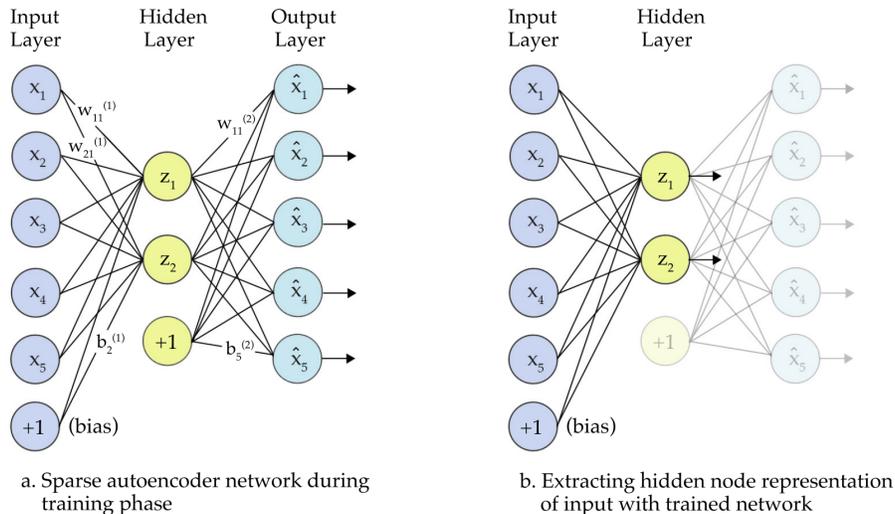
Subsequently, supervised training with fine-tuning (e.g. backpropagation) is applied to complete the training process. Unsupervised pre-training speeds up backpropagation because it allows the process to start with sensible feature detectors that are already tuned for the discrimination task at hand. This means backpropagation only needs to perform a local search from a reasonably good starting point. The unsupervised pre-training detects many features, some of which might not be useful for the discriminative task for which the model will be used. The fine-tuning process does not discover any new features; it only modifies the features slightly to correctly determine the classification boundaries.

Adding more RBM layers to the model enables the detection of progressively more abstract (deeper) features. DBNs can detect very complex and deep hierarchical features and can often provide very accurate predictions at the expense of a model that might not be conducive to human understanding. Currently, DBNs are extremely popular since they learn much faster than traditional neural networks. For example, many recent Kaggle competitions were won with DBNs. Also, Facebook recently launched a DBN research center and Google recently switched to DBNs for Android speech recognition.

## Neural Networks: Deep Autoencoders

Sparse autoencoders are typically used to deconstruct data into a compressed, distributed representation (dimensionality reduction) and then reconstruct the data (Figure 18 displays an autoencoder). In terms of structure, an autoencoder is the same as a multilayer perceptron except that there are as many outputs as there are inputs. An autoencoder (also called a denoising autoencoder) has three layers. The first hidden layer (the encoder) transforms the data into a sparse representation by converting coordinates in the input space to coordinates along the manifold. The second hidden layer (the decoder) inverts the mappings and reconstructs the input vector. When the training is complete the system has learned to map the data in both directions.

**Figure 18**



*Above: a visualization of how an autoencoder transforms inputs into a compressed representation in the first hidden layer and then reconstructs the input vector [46].*

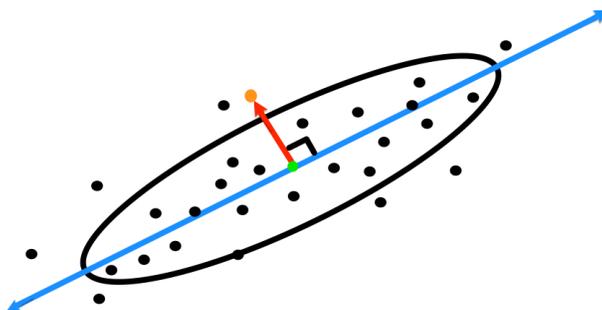
An autoencoder uses backpropagation to implement principal components analysis (PCA) in a somewhat less efficient way. The theory of PCA is that high dimensional data can often be represented in far less dimensions. This works best when data lies near a linear manifold in high dimensional space. If the manifold can be discovered, the points can be projected onto it, discarding information in directions orthogonal to the manifold. As long as there are only minor variations in the data orthogonal to the manifold, very little information is lost. Figure 19 displays an elongated Gaussian with a blue line that represents the linear manifold with the most variation. If all the points are projected onto the blue line, the information lost will be the sum of the unrepresented directions of the squared differences of the data-points from the mean.

In other words, the data was only lost in the orthogonal directions; the rest of the data was preserved. Standard principal components methods allow one to efficiently take  $n$ -dimensional data and discover the orthogonal directions that have the least variance and drop those, thus creating a more compressed version of the data. Although doing this with a neural network is less efficient, the advantage is the method can be generalized to deep neural networks where the code is a non-linear function of the output. The advantage of the non-linear approach is it gives the model the capability to handle curved manifolds in the input space.

Deep autoencoders were conceptualized in the 1980s, but due to existing training methods at the time these models could not significantly outperform PCA. In theory, encoding should be fast, but in practice it was too difficult to optimize deep autoencoders using backpropagation.

The first successful deep autoencoders were not created until 2006 when Hinton and Salakhutdinov discovered the same pre-training methods developed for deep belief networks could be applied to stacked autoencoders [54]. A deep autoencoder enables non-linear two-way mappings from the original input into a dimensionality-

**Figure 19**



Above: a two dimensional elongated Gaussian with a blue line through the direction of greatest variance. The red line is orthogonal to the linear manifold and is the direction of least variance. The distance from the green point to the orange point represents the information lost on a single training case when dimensionality reduction is applied.

reduced version and back into a reconstructed input vector.

The Hinton et al. deep autoencoder used a 784-1000-500-250-30 pattern where 784 pixels from handwritten MNIST digits were transformed through three hidden layers (with 1000, 500, and 250 neurons respectively) into a compressed encoding of 30 real numbers. The same pattern was used in reverse (with three hidden layers of 250-500-1000) to reconstruct the image. The reconstructed digits look like slightly blurry versions of the original digits with some of the minor defects missing (from the original images). This was a significant discovery because for the first time a deep autoencoder clearly outperformed PCA [54].

## Experimental Evaluation

The long term goal of this research project is to assist the University of Wisconsin-Madison Small Molecule Screening Facility (SMSF) by improving the SMSF's virtual enrichment process from 10 fold up to 40 fold or greater. As explained in earlier sections, deep learning is ideal for extracting deep hierarchical feature representations and multitask learning has demonstrated evidence of massive scalability. The ensemble of these two methods has proven effective for solving some recent bioinformatics problems. The success of various multitask deep learning experiments such as Stanford-Google MMNT [1] and Hinton's Merck Kaggle victory [48] inspired us to replicate their success by creating our system based on these algorithms.

Our plan was as follows:

1. Collect a massive amount of public bioassay data.
2. Create baselines with traditional machine learning methods.
3. Build multitask deep learning models.
4. Run experiments to confirm we can achieve the relative accuracy levels of past experiments.
5. Extend and enhance the models to provide a practical solution to the SMSF.

## The Datasets

In order to replicate past experiments we required four large public cheminformatic datasets: PCBA, Tox21, MUV, and DUD-E (Table 1). These four combined datasets contain 1.6 million compounds and 258 different targets (mostly, but not all protein). The PubChem BioAssay Database (PCBA) is the largest dataset with 5000 protein targets and 30 gene targets [55]. The 128 protein targets were selected from PCBA due to the high number of training samples in those respective datasets.

The Maximum Unbiased Validation (MUV) dataset is designed to help prevent common pitfalls in virtual screening and provides the results for 17 very challenging targets [56]. The Tox21 dataset contains an analysis

of chemicals screened against 12 targets through the federal toxicity testing of the 21<sup>st</sup> century [57]. DUD-E provides useful decoys designed to gauge the accuracy of virtual docking programs [58]. All of these datasets

**Table 1**

DATASET	TARGETS	ACTIVES	INACTIVES
Maximum Unbiased Validation (MUV)	17	510	255,000
PubChem BioAssay Database (PCBA)	128	472,440	35,631,830
Toxicology in the 21 <sup>st</sup> Century (Tox21)	12	7,121	90,944
Database of Useful Decoys: Enhanced (DUD-E)	102	40,482	1,414,421
<b>Totals</b>	<b>259</b>	<b>520,553</b>	<b>37,392,195</b>

*The table above displays a summary of the training data used in our experiments.*

provide a binary result for each compound {inactive, active} or {0, 1}, where active means the compound is likely to bind with the target and inactive means the compound is inert.

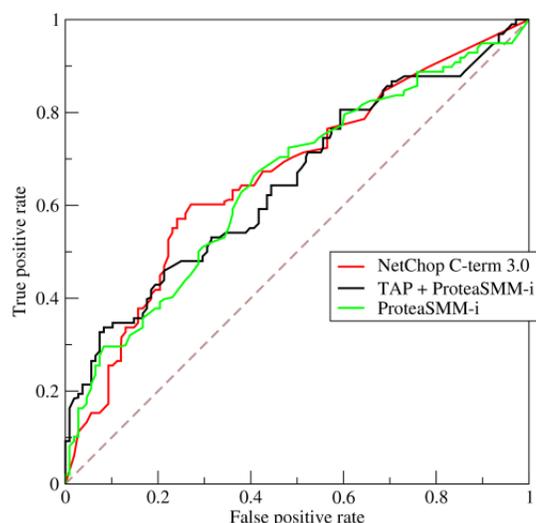
The rationale behind gathering targets from a plethora of public datasets was to amass a collection of data that could leverage the power of both deep learning and multitask algorithms. Given a set of related tasks, a multitask dataset has the potential for higher accuracy since the learner is forced to detect latent structures in the data which are true across all tasks. This makes overfitting more unlikely and provides features that might not be learnable for a single task. In order for the learner to discriminate, we converted all the chemical compounds into extended connectivity fingerprints (ECFP4). The ECFP4 fingerprinting algorithm is designed to capture features relevant to molecular activity which is ideal for drug discovery [59].

The ECFP4 algorithm assigns numeric identifiers to each atom, but not all numbers are reported. The process can be viewed as a form of lossy compression since slightly different, but very similar molecules can both be assigned the same numeric identifiers. Increasing the number of bits reported can reduce the chances of a collision, but there are also diminishing returns in the accuracy gains obtainable with longer fingerprints (e.g. 1024 bit, 2048 bit, or larger fingerprints can be used). This and computational complexity concerns were the pragmatic reasoning that resulted in our decision to use 1024 bit ECFP4 fingerprints.

It is possible an active molecule could map to the same ECFP4 fingerprint as an inactive compound which creates some risk for artificial enrichment (this could negatively impact AUC). There was also significant known overlap between the datasets as well. For example, PCBA data is the primary source of MUV data. Based on our analysis there was a degree of overlap in the datasets where distinct molecules were mapped to the same 1024 bit ECFP4 fingerprint. The exact extent of the overlap between datasets and actives or inactives may need to be explored further.

Our datasets distribution is heavily skewed towards inactive compounds: only 1.37% of the 37.9 million training examples are active (about 0.5 million). In order to prevent learning problems in our models due to the biased active-to-inactive ratio, we oversampled the active compounds and provided stratified samples to our learning models. This helped prevent false negatives and improved the overall AUC (area under curve) scores. Although recent research has shown that AUC might be an imperfect measure, it is still more robust than using a measure such as accuracy percentage.

**Figure 20**



Above: an example of an ROC curve. AUC is the area under the ROC curve [60].

For example, one could achieve 98.63% average accuracy across all models if the model was hard-coded to always predict negative (since only 1.37% of the training examples are inactive). Using AUC as our accuracy measure prevents this kind of manipulation and provides more statistically significant result. Furthermore,  $k$ -fold average AUC was necessary to contrast our benchmarks with previous work [1] (Figure 20 illustrates examples of typical AUC curves).

## Training, Validation and Test Sets

We measure the performance of each machine learning method with average AUC scores from stratified  $k$ -fold cross validation. Two files were built for active and inactive compounds respectively for each of the 259 targets. Oversampling was done in memory after the data was read from disk to provide a more balanced ratio of active to inactive training examples (and to prevent excessive bias for negative predictions). Each file contained a row with the 1024 bit ECFP4 string for the target compound, a binary {0, 1} to indicate if the target was active, and the native ID of the compound in the original dataset. Each file was divided into 5 folds, where 4 folds were used for training and 1 fold was held out for testing.

The fold ID for each training vector was determined in advance (after shuffling the data). We did this to assure each machine learning model was training and testing on the same folds to reduce the chances for inflated or deflated AUC scores for any particular learning algorithm. In the case of our deep learning models, 3 folds were used for training, and the other 2 folds were used for validation and testing respectively. The validation fold was crucial for correctly training the deep learning models. We initially attempted to run our fine-tuning validations on test data, but this consistently resulted in overtraining and poor AUC scores. Using held-out validation folds obviated this issue and resulted in higher accuracy because the early stopping mechanism was able to detect a problem and terminate training.

A separate dataset was built for training multitask learning models. Distinct multitask datasets were built for PCBA, MUV, Tox21 and DUD-E. Multitask mini-batches were created which contained data from all tasks in each respective dataset. For example, a DUD-E multitask dataset was generated using only data from DUD-E tasks, drawing equal proportions of training examples from each respective task. The same approach was used for MUV, and Tox21. In the case of PCBA, only five tasks were used (although we plan to generate a dataset for all 128 PCBA jobs in the future). In the future we will create a massively multitask dataset which includes all 259 targets.

In order to make our multitask datasets practical, we decided to include the truth set for each training example

across all datasets. To do this, one hashmap was generated for each dataset respectively (PCBA, MUV, DUD-E and Tox21) where the first column contained the 1024 bit string identifier for the compound, and subsequent columns contained the activity level for the compound of each respective target. Column identifiers (IDs) were predetermined based on the alphabetical ordering of each target name. Data was then inserted into each row with the global truth labels for each training example (which was already stored in the universal hashmap for that multitask job). The data in a given row followed this pattern where {0 , 1} indicates inactive / active for each target:

**<1024 bit string> 0 0 1 0 ... 0 1 0 0**

**Figure 21**

Model	PCBA ( $n = 128$ )	MUV ( $n = 17$ )	Tox21 ( $n = 12$ )	Sign Test CI
Logistic Regression (LR)	.801	.752	.738	[.04, .13]
Random Forest (RF)	.800	.774	.790	[.06, .16]
Single-Task Neural Net (STNN)	.795	.732	.714	[.04, .12]
Pyramidal (2000, 100) STNN (PSTNN)	.809	.745	.740	[.06, .16]
Max{LR, RF, STNN, PSTNN}	.824	.781	.790	[.12, .24]
1-Hidden (1200) Layer Multitask Neural Net (MTNN)	.842	.797	.785	[.08, .18]
Pyramidal (2000, 100) Multitask Neural Net (PMTNN)	<b>.873</b>	<b>.841</b>	<b>.818</b>	

*Above: results from the original Stanford-Google MMNT project [1]. This project report describes phase 1 of our project which was to replicate these results with our own multitask deep learning system.*

Multitask datasets were built for PCBA, MUV, Tox21, and DUD-E. The complete set of actives and inactives for each task were sampled evenly and with replacement. For example, Tox21 has 12 tasks, so each task would represent 1/12th of the training data. Mini-batches were generated where each mini-batch was stored in a file containing about 10,000 training examples, where half the examples were active and inactive respectively. Each task in the mini-batch had an equal proportion of training examples as well. We discovered that when large batches of data were loaded into memory as a matrix, this would often cause the entire server to crash. Storing 10,000 examples per file solved the matrix memory by limiting the size of a full batch (each file was 10.5mb; the files that caused the memory constraint problems were 300mb). Mini-batches were created by splitting files with 10,000 examples into sets of 100. Batch files of 10,000 examples each were produced drawing from all tasks evenly with replacement.

## Experiment Design

We chose logistic regression and random forests as our baseline single task machine learning models. These were the same baselines used in the MMNT experiments (Figure 21). We used scikit-learn to implement logistic regression, random forests and to calculate AUC. Tuning was performed on these learners to assure they performed with reasonably high AUC. The original MMNT project experimented with 10 or 20 different types of deep neural networks. We decided to only replicate the highest performing neural network architectures: the pyramidal single task neural network (P-STNN) [2000, 100] and the pyramidal multitask neural network (P-MTNN) [2000, 100]. One key difference between the MMNT experiment and our experiment is the chosen activation functions. MMNT used ReLU activations which require no pre-training. In contrast, we chose sigmoid activation units with pre-training, with plans to use ReLU activations instead if we encountered sub-par AUC results.

A “pyramidal” neural network refers to a multi-layer feed-forward network which expands wider in the first layer, eventually pushing into a very narrow final layer (i.e. roughly in the shape of a pyramid). The narrow final layer

then feeds into a single threshold function for single task neural networks or a softmax layer for multitask neural networks. In the next section we will describe our deep neural network design in more detail.

## Pyramidal Deep Belief Networks: Multi and Single Task

The Stanford-Google MMNT experiment was not clear on which deep learning algorithm MMNT used. Both deep autoencoders and deep belief networks (DBNs) are viable candidates and both have similar architecture (as explained above in earlier sections of this report). We chose to implement Deep Belief Networks with plans to possibly explore deep autoencoders after the summer-term project (which is beyond the scope of this report). Since our plan was to deliver a practical solution with extremely high AUC, we decided we should use an existing library to implement our DBN-based P-STNNs and P-MTNNs.

Building high performance, computationally efficient deep learning systems is a non-trivial task. Hence, deep learning is completely different from traditional machine learning algorithms in both computational complexity and system architecture. At the time of this writing deep learning is not available in standard libraries such as Weka, scikit-learn, and Orange. However, there are a large number of specialized deep learning libraries, three of which were potentially compatible with our need to massively scale: Torch (Facebook), Theano (community), and DL4J (Skymind). Daily commits and bug-fixes from many contributors and other indicators suggest Theano has a strong vibrant community, actively extending and improving the codebase. After careful deliberation Theano was chosen due to the combination of excellent documentation, high performance (Nvidia CUDA compatibility for massive parallel GPU-based computations), community support, and high activity on Github.

Theano has a difficult learning curve and generating our own custom models based on this library was a non-trivial task. We designed our DBN pyramidal single task deep belief networks (P-STNN) using Theano with the following layer pattern: 1024-2000-100. The input layer feeds into a hidden layer (h1) with 2000 sigmoid neurons and then the next hidden layer (h2) compressed the output into 100 neurons. One key difference with our experiments is we used Sigmoid activations (instead of ReLU). After initializing random weights, greedy, layer-wise unsupervised pre-training with contrastive divergence was applied to bring the neural network to an energy equilibrium (sigmoid-based architectures require pre-training). Backpropagation with stochastic gradient descent (SGD) was then used to fine-tune the output to make classifications based on the discovered features that were most significant for our task. As with the baselines, scikit-learn utilities were used to calculate the AUC results.

## Experiment Results

The pyramidal multitask neural network [2000, 100] (P-MTNN) part of our project is still in progress and completion is not expected until later in August 2015. Computational resources have continuously been a major bottleneck in our experiment since our dataset is truly “massive” (40GB) and deep learning has substantial computational requirements. Iterating through experiments, and determining the right settings has been a monumental ask at times and we could not have produced results without the generous assistance of the Simulation Based Engineering Laboratory (SBEL) and HTCondor. To an extent, the benefit of automatic feature

**Table 2**

	PCBA	MUV	Tox21	Overall
Logistic Regression (LR)	0.81	0.77	0.80	0.79
Random Forests (RF)	0.78	0.68	0.85	0.73
P-STNN [2000, 100]	0.84	0.80	0.82	0.82

*LR, RF, and P-STNN AUC results were very similar to those of the MTNN experiments, except for P-STNN PCBA. Notably, our 1 k-fold of P-STNN outperformed the MTNN experiments.*

detection is offset by the difficulty in finding the right training settings.

Aside from P-MTNN, the rest of our experiments are complete with one exception. Our 259 single task P-STNNs only ran on a single fold configuration due to time and computational complexity limitations. Our final 5-fold P-STNN results may vary slightly from these initial results. Based on experience and the large number of tasks (259), it is unlikely the results will vary significantly from our present findings.

Although our random forests had spotty performance, our logistic regression and P-STNN results substantially outperformed the Stanford-Google MMNT results (compare Figure 21 to Table 2). We also matched the MMNT results with DUD-E since all our baselines and P-STNNs achieved 0.99 AUC with that dataset. Since DUD-E provides no point of differentiation between our machine learning methods, or between our project and the MMNT experiment, it is not reported in the table. Logistic regression was not a focal point for us, and we do not know why this baseline outperformed MMNT by such a large margin. Further investigation is required to ascertain the reasons for this.

The P-STNNs from the MMNT experiment performed roughly the same as the logistic regression (LR) and random forest (RF) baselines. In fact, only the final multitask neural network results (P-MTNNs) outperformed the best baselines. We were pleasantly surprised that our DBN-based P-STNNs outperformed all the other baselines achieving 0.82 AUC overall compared to RF at 0.73 and LR at 0.79. Random forests slightly outperformed our P-STNNs on the smallest dataset Tox21 with 0.82 AUC which is 0.03 higher than P-STNNs.

Why did our single pyramidal deep belief networks outperform the MMNT single task deep networks? One possible reason is the MMNT experiments omitted pre-training. As explained above, our experiment used pre-training, a powerful regularizer that helps avoid local minima and prevents over-training. In contrast, the only regularizer used in the MMNT experiments was 0.25 dropout in both the P-STNNs and P-MTNNs (most likely early stopping was used as well, although this is not mentioned in the MMNT paper). While pre-training is not 'necessary' in training ReLU-based architectures [71, 72], this does beg the question. It is also possible that sigmoid activations are more conducive to this problem. However, many claim ReLU units are more realistic for biological systems.

There is another possible reason for the performance discrepancy: the ultimate focus of MMNT was not P-STNNs; instead, the focus was on proving multitask learning can scale in a drug discovery context. In contrast, our focus is to leverage both the power of deep learning and multitask algorithms to maximize the potential for improved enrichment. More investigation is required for us to understand the performance differences, although perhaps it is too early for a final conclusion (it is possible that our performance discrepancies will disappear once we run the remaining folds).

## **Achieving Higher Performance With DBNs**

While training our deep belief networks, we frequently experienced problems where too many neurons were set to "off" positions (zeroed out) and the network could only predict negative results (resulting in 0.50 AUC). This often occurred when the pre-training learning rate was set too high. Inherently, the "greedy" part of unsupervised pre-training is an indicator that a powerful learning method is being used. Greedily applying layer-wise contrastive divergence to bring each of the respective layers to an equilibrium is vastly more aggressive than the fine-tuning process of SGD. This is somewhat common knowledge to an extent. As evidence, the Theano default settings use a pre-training learning rate of 0.01 and a fine-tuning learning rate of 0.1 (the pre-training rate is  $1/10^{\text{th}}$  the fine-tuning rate). The default pre-training learning rate was completely wrong for our experiments, and a number of iterations were required to discover more ideal settings.

When predicting protein-ligand binding in the context of PCBA, MUV especially, a gentle pre-training rate must be used to avoid breaking the model. We found that just 10 epochs at the absurdly low pre-training learning rate of 0.0000003 was gentle enough to never break our P-STNN DBNs. This might not be the ideal number of pre-

training epochs but 10 epochs at the low learning rate worked well enough. These settings were the parameters in our best P-STNN performance.

In future experiments it would be interesting to see how the same learning rate would perform with 100 or 1,000 epochs of pre-training on the various datasets. Also, at least with our experiments, pre-training was a necessity; when we attempted to bypass this step, our DBNs generalized poorly. In contrast to pre-training, the fine-tune learning rate for SGD was not nearly as sensitive. The rate of 0.05 worked pretty well, and is not so distant from the default Theano setting of 0.1. In future experiments, we may try more optimizations such as weight decay, dropout and Nesterov momentum.

## Conclusions

Our goal was to explore the power of multitask learning combined with deep neural networks. Our single task baselines and DBNs (P-STNN [2000, 100]) performed well. However, we still need to obtain *multitask* DBN (P-MTNN [2000, 100]) results; this is critical in order for us to validate our hypothesis. Substantial progress was made on the project. The bulk of the work is now complete including data collection, cleaning and preparation for multitask and single task models, as well as creating and testing the various machine learning algorithms required to complete our experiments. We have also achieved strong performance results with our single task deep belief networks, which marks an important progress milestone.

In terms of system architecture, the P-MTNN is very close to the P-STNN design. The key difference is the P-MTNN must learn more slowly since it must absorb an order of magnitude more training examples from diverse training sets. We have built a strong foundation for this experiment by creating accurate P-STNN learners and accurate baselines. We have also improved our depth of knowledge and experience with deep learning. Notably, we made a few interesting discoveries regarding ideal learning rates for pre-training the public datasets PCBA, DUD-E, Tox21 and MUV (something not discussed by the MTNN project since they did not need pre-training).

Many aspects of the deep learning experiments were non-trivial to set up since little is published on the practical details that are often required with bleeding edge technology. Once we obtain our final P-MTNN results we can begin the search for ways to apply our findings towards improving the SMSF's enrichment process. These improvements may include ensemble methods that combine existing enrichment approaches such as AutoDock Vina with deep learning, or we could add more training data to our system from public sources like ChEMBL to see if we can determine the point where multitask benefits reach a plateau.

Michael Newton and his UW Madison research team are also working concurrently on ways of clustering compounds and proteins to discover shared latent properties. Cluster-based multitask job amalgamations could potentially result in better predictions for protein-ligand binding activity. We could also extend this project to combine fingerprints and bioactivity features in ways that would allow us to extend the multitask deep belief network to make predictions on new unknown targets which are not in the existing datasets.

## References

- [1] Ramsundar, Bharath, et al. "Massively Multitask Networks for Drug Discovery." *arXiv preprint arXiv: 1502.02072* (2015).
- [2] Scannell, Jack W., et al. "Diagnosing the decline in pharmaceutical R&D efficiency." *Nature reviews Drug discovery* 11.3 (2012): 191-200.
- [3] "Plate Reader." Wikipedia: The Free Encyclopedia. Wikimedia Foundation, Inc. 22 July 2015. Web. 22 Jul. 2015. <[https://en.wikipedia.org/wiki/Plate\\_reader](https://en.wikipedia.org/wiki/Plate_reader)>
- [4] Storrs, Carina. "Screening Goes In Silico. Computational Tools Take Some of the Cost and Guesswork out of Drug Discovery." *The-scientist.com*. 1 Feb. 2015. Web. 23 July 2015. <<http://www.the-scientist.com/?articles.view/articleNo/41979/title/Screening-Goes-In-Silico/>>
- [5] Wikipedia, the free encyclopedia. Chromosome. Available at: <https://en.wikipedia.org/wiki/Chromosome>. Accessed July 31, 2015].
- [6] Wikipedia, the free encyclopedia. Protein. Available at: <https://en.wikipedia.org/wiki/Protein>. Accessed July 31, 2015].
- [7] Simoni, Robert D., Robert L. Hill, and Martha Vaughan. "Urease, the first crystalline enzyme and the proof that enzymes are proteins: the work of James B. Sumner." *Journal of Biological Chemistry* 277.35 (2002): e23-e23.
- [8] Griffith, Fred. "The significance of pneumococcal types." *Journal of Hygiene* 27.02 (1928): 113-159.
- [9] Watson, James D., and F. H. Crick. "Molecular structure of nucleic acids: a structure for deoxyribose nucleic acid. JD Watson and FHC Crick. Published in Nature, number 4356 April 25, 1953." *Nature* 248.5451 (1974): 765.
- [10] Zadeh, Lotfi A. "Fuzzy sets." *Information and control* 8.3 (1965): 338-353. <<http://www.cs.berkeley.edu/~zadeh/papers/Fuzzy%20Sets-Information%20and%20Control-1965.pdf>>
- [11] Woolf, Peter J., and Yixin Wang. "A fuzzy logic approach to analyzing gene expression data." *Physiological Genomics* 3.1 (2000): 9-15.
- [12] Karabegovic, Almir, Zikrija Avdagic, and Mirza Ponjavic. *Applications of Fuzzy Logic in Geographic Information Systems for Multiple Criteria Decision Making*. na, 2006.
- [13] "Molecular Therapeutics Initiative", 22 July 2015. Web. 22 Jul. 2015. <[http://www.its.utmb.edu/molecular\\_therapeutics\\_initiative/virtual\\_screening.html](http://www.its.utmb.edu/molecular_therapeutics_initiative/virtual_screening.html)>
- [14] "DNA-based methods for Bioaerosol Analysis" 24 March 2014. Web. 22 Jul. 2015. <<http://microbe.net/2014/03/24/dna-based-methods-for-bioaerosol-analysis-great-resource/>>
- [15] Miller, Kenneth R., and Joseph S. Levine. *Prentice Hall biology*. Upper Saddle River, N.J.: Pearson/Prentice Hall, 2008. Print.
- [16] Thieffry, Denis, and Sahotra Sarkar. "Forty years under the central dogma." *Trends in biochemical sciences* 23.8 (1998): 312-316.
- [17] Unterthiner, Thomas, et al. "Deep learning as an opportunity in virtual screening." *Deep Learning and Representation Learning Workshop, NIPS*. 2014.
- [18] "Merck Molecular Activity Challenge". 16 October 2012. Web. 22 Jul. 2015. <<https://www.kaggle.com/c/MerckActivity>>
- [19] Floreano, Dario, and Laurent Keller. "Evolution of adaptive behaviour in robots by means of Darwinian selection." *PLoS Biol* 8.1 (2010): e1000292. <<http://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.1000292>>
- [20] McCartney, Scott. *ENIAC: The triumphs and tragedies of the world's first computer*. Walker & Company, 1999.
- [21] Castelfranchi, Cristiano. "Alan Turing's "Computing Machinery and Intelligence"." *Topoi* 32.2 (2013): 293-299.
- [22] Turing, Alan M. "Intelligent Machinery, A Heretical Theory." *The Turing Test: Verbal Behavior as the Hallmark of Intelligence* (1948): 105.
- [23] Teuscher, Christof. *Turing's connectionism: an investigation of neural network architectures*. Springer Science & Business Media, 2012.
- [24] Wikipedia, the free encyclopedia. History of artificial intelligence. Available at: [https://en.wikipedia.org/wiki/History\\_of\\_artificial\\_intelligence](https://en.wikipedia.org/wiki/History_of_artificial_intelligence). Accessed July 31, 2015].
- [25] Rosenblatt, Frank. "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review* 65.6 (1958): 386.
- [26] Wikipedia, the free encyclopedia. Timeline of artificial intelligence. Available at: [https://en.wikipedia.org/wiki/Timeline\\_of\\_artificial\\_intelligence](https://en.wikipedia.org/wiki/Timeline_of_artificial_intelligence). Accessed July 31, 2015].
- [27] Warwick, Kevin, and Hemal Shah. "Good machine performance in Turing's imitation game." *Computational Intelligence and AI in Games, IEEE Transactions on* 6.3 (2014): 289-299.
- [28] Minsky, M. "Paper, S.: Perceptrons: An introduction to Computational Geometry." (1969).
- [29] Rumelhart, D. E. "David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams." *Nature* 323 (1986): 533-536.
- [30] McCarthy, John. "What is artificial intelligence." URL: <http://www-formal.stanford.edu/jmc/whatisai/> (2007): 38.
- [31] John MCarthy. Available at: <http://www.independent.co.uk/news/obituaries/john-mccarthy-computer-scientist-known-as-the-father-of-ai-6255307.html>. Accessed July 31, 2015.
- [32] Alan Turing. Available at: <http://www.independent.co.uk/news/people/news/alan-turing-gets-royal-pardon-for-gross-indecency--61-years-after-he->

poisoned-himself-9023116.html. Accessed July 31, 2015.

[33] Winston, Patrick H. Artificial intelligence. Reading, Mass: Addison-Wesley Pub. Co, 1977. Print.

[34] Muggleton, Stephen. "Inductive logic programming." New generation computing 8.4 (1991): 295-318.

[35] Zelle, John M., Raymond J. Mooney, and Joshua B. Konvisser. "Combining top-down and bottom-up techniques in inductive logic programming." Proceedings of the Eleventh International Conference on Machine Learning. 1994.

[36] Wong, Man Leung, and Kwong Sak Leung. Data mining using grammar based genetic programming and applications. Vol. 3. Springer Science & Business Media, 2006.

[37] Machine Learning Cheat Sheet. Available at: <http://peekaboo-vision.blogspot.com/2013/01/machine-learning-cheat-sheet-for-scikit.html>. Accessed July 31, 2015.

[38] Random Forests. Available at: <http://blog.yhathq.com/posts/random-forests-in-python.html>. Accessed July 31, 2015.

[39] Random Forests. Available at: [http://blogs.sas.com/content/analitika/files/2012/04/blog\\_4\\_11.jpg](http://blogs.sas.com/content/analitika/files/2012/04/blog_4_11.jpg). Accessed July 31, 2015.

[40] Breiman, Leo. "Random forests." Machine learning 45.1 (2001): 5-32.

[41] Quinlan, J. Ross. "Induction of decision trees." Machine learning 1.1 (1986): 81-106.

[42] Perceptron. Available at: <http://tex.stackexchange.com/questions/104334/tikz-diagram-of-a-perceptron>. Accessed July 31, 2015.

[43] Single layer neural network Available at: <http://igva2012.wikispaces.asu.edu/aarvay - Paper 3 - Data driven visualization of neural networks>. Accessed July 31, 2015.

[44] Hinton, Geoffrey E., Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets." Neural computation 18.7 (2006): 1527-1554.

[45] Deep Belief Networks. Available at: <http://www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/DeepBeliefNetworks>. Accessed July 31, 2015.

[46] Autoencoder. Available at: <http://www.cs.cmu.edu/~afyshe/papers/SparseAE>. Accessed July 31, 2015.

[47] McCulloch, Warren S., and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity." The bulletin of mathematical biophysics 5.4 (1943): 115-133.

[48] Dahl, George E., Navdeep Jaitly, and Ruslan Salakhutdinov. "Multi-task neural networks for QSAR predictions." arXiv preprint arXiv:1406.1231 (2014).

[49] Wikipedia, the free encyclopedia. In silico. Available at: [https://en.wikipedia.org/wiki/In\\_silico](https://en.wikipedia.org/wiki/In_silico). Accessed July 31, 2015].

[50] LeCun, Yann, and Yoshua Bengio. "Convolutional networks for images, speech, and time series." The handbook of brain theory and neural networks 3361.10 (1995).

[51] Deep Learning image Available at: <http://www.rsipvision.com/exploring-deep-learning/>. Accessed July 31, 2015.

[52] Erhan, Dumitru, et al. "Why does unsupervised pre-training help deep learning?." The Journal of Machine Learning Research 11 (2010): 625-660.

[53] Ackley, David H., Geoffrey E. Hinton, and Terrence J. Sejnowski. "A learning algorithm for boltzmann machines\*." Cognitive science 9.1 (1985): 147-169.

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

[55] PubChem's BioAssay Database (PCBA) <http://www.ncbi.nlm.nih.gov/pubmed/22140110>

[56] Rohrer, Sebastian G., and Knut Baumann. "Maximum unbiased validation (MUV) data sets for virtual screening based on PubChem bioactivity data." Journal of chemical information and modeling 49.2 (2009): 169-184.

[57] The Tox21 Dataset <http://www.epa.gov/ncct/toxcast/data.html>

[58] Mysinger, Michael M., et al. "Directory of useful decoys, enhanced (DUD-E): better ligands and decoys for better benchmarking." Journal of medicinal chemistry 55.14 (2012): 6582-6594.

[59] Rogers, David, and Mathew Hahn. "Extended-connectivity fingerprints." Journal of chemical information and modeling 50.5 (2010): 742-754.

[60] ROC curve image. Wikipedia, the free encyclopedia. Receiver operating characteristic. Available at: [https://en.wikipedia.org/wiki/Receiver\\_operating\\_characteristic](https://en.wikipedia.org/wiki/Receiver_operating_characteristic). Accessed July 31, 2015].

[61] The Central Dogma of Biology Available at: <http://microbe.net/2014/03/24/dna-based-methods-for-bioaerosol-analysis-great-resource/>. Accessed July 31, 2015.

[62] Dawes TD, Turincio R, Jones SW, et al. Compound Transfer by Acoustic Droplet Ejection Promotes Quality and Efficiency in Ultra-High-Throughput Screening Campaigns. J Lab Autom. 2015;

[63] The Pivot Park Screening Center. Available at: <http://www.pivotparkscreeningcentre.com/22/uhts-lab/>. Accessed July 31, 2015.

[64] Newton, Michael A., and Lisa M. Chung. "Gamma-based clustering via ordered means with application to gene-expression analysis." *Annals of statistics* 38.6 (2010): 3217.

[65] Hecht, David, and Gary B. Fogel. "Computational intelligence methods for docking scores." *Current Computer-Aided Drug Design* 5.1 (2009): 56-68.

[66] Morris, Garrett M., et al. "Automated docking using a Lamarckian genetic algorithm and an empirical binding free energy function." *Journal of computational chemistry* 19.14 (1998): 1639-1662.

[67] David Page, et al. "An inductive logic programming approach to validate hexose binding biochemical knowledge." *Inductive Logic Programming*. Springer Berlin Heidelberg, 2010. 149-165.

[68] Jain, Siddhartha, Anthony Gitter, and Ziv Bar-Joseph. "Multitask Learning of Signaling and Regulatory Networks with Application to Studying Human Response to Flu." *PLoS computational biology* 10.12 (2014): e1003943.

[69] Netflix. "Distributed Neural Networks with GPUs in the AWS Cloud". Available at: <http://techblog.netflix.com/2014/02/distributed-neural-networks-with-gpus.html>. Accessed August 2, 2015.

[70] Srivastava, Nitish, et al. "Dropout: A simple way to prevent neural networks from overfitting." *The Journal of Machine Learning Research* 15.1 (2014): 1929-1958.

[71] Available at: <http://fastml.com/deep-learning-these-days/>. Accessed August 3, 2015.

[72] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." *Nature* 521.7553 (2015): 436-444.