1-1-2019

Applications of Sparse Representations

Pulin Agrawal

APPLICATIONS OF SPARSE REPRESENTATIONS

by

Pulin Agrawal

A Dissertation

Submitted in Partial Fulfillment of the

Requirement for the Degree of

Doctor of Philosophy

Major: Computer Science

The University of Memphis

May 2019

Acknowledgements

I would like to thank and express my gratitude first and foremost to two of my advisors Dr. Stan Franklin and Dr. Bernie J. Daigle, Jr. for their invaluable advice and guidance over the years. I would like to thank them both for giving me the freedom to pursue my interests in research and guiding me in all my academic and professional endeavors. I would like to thank my advisory committee members Dr. Vasile Rus and Dr. Deepak Venugopal next, for serving on my committee and helping me with valuable discussions and comments on my research.

I am deeply grateful to everyone in the two research groups that I worked in, the Cognitive Computing Research Group and the Daigle Lab, for welcoming me with so much warmth and providing a friendly atmosphere that enabled all this work. A special mention to Sean Kugele for being there as friend and mentor whenever I needed either. Thanks to Dr. Javier Snaider for giving me valuable ideas that helped in completing one of the projects in this dissertation.

Finally, I am forever grateful and extremely lucky to have such an amazing family, including my aunt and uncle. My parents gave me all the freedom to follow my dreams and ambitions. Without their support I would not have been able to embark this journey or finish it. Thanks to my grandmother for her words of wisdom in dire time of need, “You cannot hope to make the best decision, what you can do is, do your best”. I am also thankful to my friends, who are no different from family, Ravi Agrawal and Rashmi Bansal, who provided me a very comfortable retreat at their place and a mentally stimulating environment that enabled me to write a large chunk of this dissertation. Thanks to my sisters Anshika, Palak and Radhika for their love and affection that kept me whole and functioning this entire time.

Preface

This dissertation explores the use of sparse representations in for encoding information for use in applications like Artificial Intelligence and Machine Learning. I believe if we want to build truly intelligent machines, we should look no further than inside our brains. Our brains use sparse representations for making sense of complex signals in our environment and enables us to take intelligent actions. Therefore, it is important to explore how sparse representations can be used for representing various kinds of information, so that we can use the expressive power of sparse representations for building truly intelligent machines.

Chapter 2 of this dissertation was accepted at the Sixth Annual Conference on Advances in Cognitive Systems in 2018 held at Stanford University, Stanford, California.

Abstract

In this dissertation I explore the properties and uses of sparse representations. Sparse representations use high dimensional binary vectors for representing information. They have many properties which make this representation useful for applications involving pattern recognition in highly noisy and complex environments. Sparse representations have a very high capacity. A typical sparse representation vector has a capacity of 10^84 distinct vectors, which is more than the number of atoms in the universe. Sparse representations are highly noise robust. They can tolerate even up to 50% noise. A very powerful and useful property of sparse representations is that they allow us to easily measure similarity between two things by directly comparing their representations. These properties allow them to have applications in a variety of fields, like Artificial Intelligence and Molecular Biology, that need to encode information that is complex and noisy in nature. In this dissertation, I show how sparse representations can be used for representing complex environments for an agent based on Learning Intelligent Decision Agent (LIDA) model. Sparse representations allowed us to achieve a two-fold goal of producing information rich representations of things in the environment while proposing a method of generating grounded representations for the LIDA model. Sparse representations also allowed us to ground the representations used by LIDA in the sensory apparatus of the agent while still allowing a perfect fidelity communication between the sensory memory of LIDA and the rest of the model. I also show how sparse representations are useful in Molecular Biology for discovering data-driven patterns in heterogeneous and noisy gene expression data. We used a sparse auto-encoder to learn sparse representations of transcriptomics experiments taken from a huge publicly available dataset. These representations were then used to identify biological patterns in the form of gene sets. The representation provided a unique signature for a set of

samples originating from the same experimental condition. Applications of our method include the identification of previously undiscovered gene sets as well as supervised classification of samples from different biological classes. Overall, our results show that sparse representations are useful in a variety of fields that involve finding patterns in a complex and noisy environment.

Table of Contents

List of Tables

List of Figures

CHAPTER 1: Sparse Representations: Why they deserve attention?

1. Introduction

One of the first things one encounters when dealing with information is its representation. Some well-known methods that information is represented for humans is words and bytes for computers. They are useful because the processing machines for which they are made can recognize information encoded in this format and process it efficiently in various ways to produce knowledge.

It is very important to represent information in a way which makes its processing fast and easy. A human can learn to read a byte (8-bits) at a time and process information using that, but it is not easy or fast to read and process bytes for humans. Similarly, it took a whole field of research of Natural Language Processing (NLP) and multiple years to make a computer program understand words, and yet not perfectly. Another example of this shows up in Japanese writing system. Japanese writing system does not have spaces. Therefore, it can be extremely hard to read Japanese. Kanji makes it so that Japanese becomes easily readable. Kanji are Chinese characters that are borrowed in Japanese writing system. They replace most nouns and some verbs. Other joining words are written in native Japanese writing system i.e. Hiragana.

For example,

One can write “let’s go home” using native Hiragana system as follows:

うちにかえろう。

But it is difficult to read because it is hard to figure out where one word ends and the other begins. Alternatively, it is written in the following way, which is how Japanese would write it using Kanji.

家に帰るう。

It is relatively easier to figure out that 家, に and 帰 are individual components. Even for a non-native reader it is simpler once they get to know that Kanji is generally more complex in terms of number of strokes than Hiragana. So, 家 (house) and 帰 (go) would be Kanji and に, る and う would be Hiragana.

Thus, it becomes important to adapt the way of representing information such that it facilitates the task at hand. Winston has suggested that, “Once a problem is described using an appropriate representation, the problem is almost solved” (1992, p. 218). Using correct information representation has helped many fields in solving problems. Kanerva (2009) describes how one representation can be better suited for one application but very ill-suited for others. For example, a binary search tree is a great way to store information for searching but a bad way to store for memory management. Franklin (1997) has discussed that representations play a very important role for both symbolic and connectionist Artificial Intelligence (AI) models.

2. Representing Information

Every field employs methods of information representation that suit the kinds of processing done in that field. In the following sections I discuss methods of representing information used in the two major fields that are relevant to this dissertation: AI and molecular biology.

2.1. Representing Information in AI

Some early AI systems also used Search Trees to represent the of states of a system. First Order Logic rules were also among the first to be used in AI and have seen decades of research but have proven to be of limited use. Other common methods of knowledge representation are frames (Minsky, 1974), ontologies and semantic nets. A very common form of representing information in AI systems is using semantic nets. These networks are symbolic representations of concepts learned by the system in the form of nodes and links. Nodes represents concepts and links represents relationships between these concepts. An example of that is the copycat architecture (Hofstadter & Mitchell, 1994). This representation is easy to understand and navigate for humans but is not the best for a computer system. This is because graph structures become difficult to navigate due to the presence of cycles and difficult to understand which nodes are conceptually similar based on all the links. More recently connectionist representations have gained more traction due to availability of better computational power. It has become feasible to train huge neural networks. Here information is represented by activations of mostly unstructured collection of units. These are commonly generated with the help of neural networks. Representations in such systems arise in ways like brains. The units are connected with the inputs using links that model synapses of neurons or other units that model neurons with links. Input activations are passed through the network of these units and controlled

by some form of training rules, which modifies the weights of the links. This effects in modulating the activations of the units. In fact, sparse representation is an example of a connectionist representation, as we will see in Section 7 (Methods of Producing Sparse Representations). There is an ongoing debate on which type of representation are more suited for AI systems (Dinsmore, 2014; Smolensky, 1987).

2.2. Representing Information in Molecular Biology

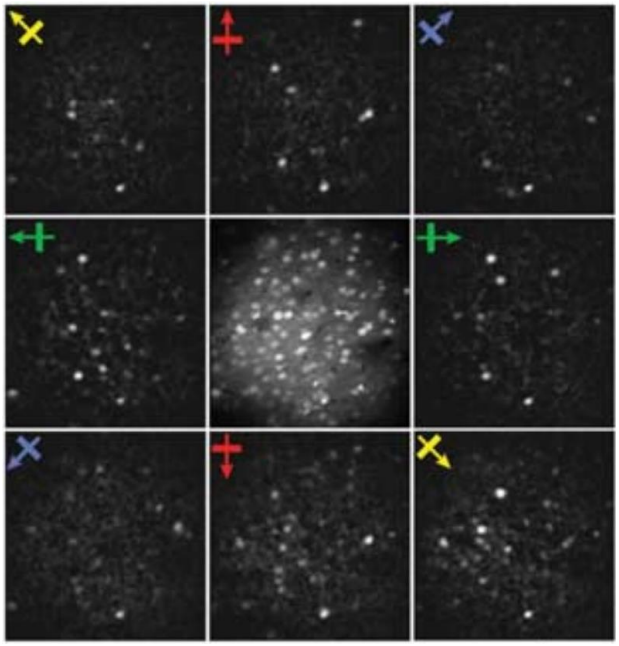
High-throughput experimental studies in molecular biology represent data in a few different ways. Raw data are often in the form of a matrix of samples by (thousands of) gene expression values. Given the high-dimensionality of the data, gene set collections such as Gene Ontology (Ashburner et al., 2000) biological processes (BP) enable biological interpretation of particular gene expression patterns. Matrix factorization methods (Daigle Jr et al., 2010; Taroni et al., 2018) have been used to represent data in the form of latent variables that are linear combinations of gene expression. Some recent methods (Tan, Doing, et al., 2017; Tan, Huyck, et al., 2017) of analysis that used neural network models would represent information in the form of activations of units in the network as described above. Unit activations on neural networks are typically nonlinear combinations of input, i.e. gene expression values here.

In AI systems, data comes from some form of sensors in an environment. They can be visual, auditory or proprioceptive sensors in the case of robots, but they can also be text or databases in case of software agents. In Molecular Biology, gene expression data often comes from DNA micro-array platforms. These platforms are made up of thousands of DNA probes that measure things like concentration of a specific RNA or DNA sequence in a sample. Both of these systems have a very complex range of raw data. Both these fields can use help from a way of

representation that can deal with complex data. I believe sparse representation can be of use in both these fields. Even evolution seems to have stumbled upon the benefits of sparse representations to represent complex information about the environment in biological brains.

3. Brain Using Sparse Representation

It has been suggested that the function of a region or a macro-column in the brain is to store sparse distributed representation of its inputs and act as a recognizer for it in the future (George & Hawkins, 2005; Rinkus, 2014). The amount of evidence for sparse representations being used for this in brains is rapidly increasing. (Ohki, Chung, Ch'ng, Kara, & Reid, 2005) showed calcium imaging results from a region of V1 in the neocortex when oriented bars were shown in a visual stimulus. The calcium imaging shows that, in an area of brain that has approximately 3,000 neurons, less than 10 of them showed activity for one oriented bar as visual stimulus, as shown in Figure 1.



(Ohki et al., 2005)

Figure 1 Calcium imaging of a cross section of cortical column shows as few as tens of neurons getting active out of about 3000 neurons, as shown in each of the boxes on the periphery, for an input stimulus of an oriented bar. The center image represents the average activity of the cells for all the oriented bars.

Olshaulsen and Field (2004) suggest four reasons why it is better to use sparse representations for the brain. First, it allows for increased storage capacity. This will also be shown to be true in Section 6.2. Second, it relates closely to the structure in the natural images. Third, the representation becomes very specific to the presence of certain features in the image and thus can be used for learning higher order features efficiently. Fourth, and a very strong evidence in favor, is that it saves energy. This a huge evolutionary advantage to biological creatures. Sparse representations are able to save energy because only a few out of the billions of neurons will be active in representing something, which spares a large portion of the brain from using energy. In

my opinion, this is probably the reason why brains can perform such complicated processing with only a fraction of power that it takes for modern day computers.

It was also shown by Olshaulsen and Field (1996) that if we try to learn sparse coding for representing natural images such that the representation allows faithful reconstruction of the input, we end up with receptive fields that are found in simple cells of the visual cortex. This strongly suggests that the underlying mechanism that the brain uses to represent information is by trying to produce a sparse code of the input.

4. Sparse Representation in AI

It appears sparse representations is a useful way of representing information for biological brains. It offers a large memory capacity in creatures like humans and enables us to perform a wide variety of functions with far more efficiency than computers can. It then stands to reason that sparse representations are suited for the kinds of processing that the brain does. like identifying closely related things as similar while still being able to function in a highly noisy environment. So, when we try to emulate brain-like functions as AI techniques, sparse representations can prove to be very helpful.

I propose that sparse representations are most useful in places where the nature of the information that is needed to be represented is very complex. Therefore, sparse representations would be especially suited to Artificial Intelligence (AI) applications because AI is generally used in places where the nature of information is complex. Complexity could be due to high number of variables like in system monitoring or due to high degree of semantics packed into

small number of variables like a chess playing agent. As a tradeoff, if the nature of information is simpler the cost of computation of sparse representations might overwhelm the benefits of using it.

5. Sparse Representations: A Background

5.1. What is sparse representation?

Sparse representations are a very useful way of representing complex information, using what is called an overcomplete basis set. An overcomplete basis set is a set of basis vectors where, the number of basis vectors is more than the dimensionality of the input. In other words, when we use sparse representation, we map the input into a higher dimensional space. However, to rule out a trivial map into a higher dimension there is a constraint on this mapping. The constraint is that for any given representation only a very small percentage of basis vectors are used, much smaller than the dimensionality of the input. For a system using these representations, the number of basis vectors that are used remain relatively constant.

These basis vectors can also be thought of as latent features. Therefore, an alternate way of understanding a sparse representation is that it is all the features present in an input out of a dictionary of all possible features in that input domain. It is therefore in the class of problems of dictionary learning, where there is a constraint of sparsity on the representation from the dictionary for an input.

There can be many ways to do such a mapping or dictionary learning. Thus, there are various algorithms for producing sparse representations. Some of them are discussed in Section 7. This

form of representation is also referred to by slightly different names. Other names that are used are sparse coding, sparse distributed representations or sparse dictionary learning.

Sparse representations are most commonly seen in terms of high dimensional binary vectors. These have a very few percent of bits on, usually between 2%-5%, for one representation. They are also called sparse distributed representations because the on bits are distributed across the vector and there is no spatial localization enforced on it. For example, we could represent a 'man' by a binary vector as follows,

man= [00000011000000000000000000010000000010000000000000]

Now if we want to represent the concept of a 'tall man' sparse representation allows us to have a binary representation of a 'tall man' as follows,

tall_man= [00000011000000000000000000010000000000000000000001]

Now the 'tall man' vector is very similar to the man vector, except for one bit, so a tall man can act like a 'man' vector, for all practical purposes, but it can demonstrate the property of being tall by the extra feature that is on. We will show in later sections how sparse representations are able to demonstrate this property of similarity.

5.2. A Brief History of Sparse Representations

It is suggested by some (Ahmad & Hawkins, 2015; Zhang, Xu, Yang, Li, & Zhang, 2015) that (Donoho, 2006) was the first to propose sparse coding in the form of compressed sensing in 2007. However, it seems that the mathematical foundations of sparse coding were laid much earlier by Olshaulsen and Field in 1996 (Olshausen & Field, 1996). They tried to account for the properties of simple-cell receptive fields by trying to maximize the sparsity of code for natural images. Also, many concepts relevant to sparse coding were also present in Kanerva's (Kanerva, 1988) proposal of Sparse distributed memory when he described the properties of high dimensional spaces. However, Kanerva's description is about a memory model for storing binary vectors and not for representation.

5.3. Sparse Representations and Sparse Distributed Memory

There have been descriptions of Sparse Representation and Sparse Distributed Memory (SDM) as two concepts that demonstrate ways in which brain stores representations. To my knowledge no other description resolves the two as comparable ideas. Sparse Representation is a method of representing information in the form of high dimensional binary (usually) vectors. SDM is a method of storing high dimensional dense binary vectors, often called words in this context, in a sparse set of points in a high dimensional space. SDM is created out of this finite set of sparsely distributed locations, called hard locations. A binary vector (or a word) is stored in a subset of these hard locations, which can later be retrieved using a partial or noisy form of this binary vector. Sparse Representation and Sparse Distributed Memory are two aspects of the same concept related to hyperdimensional computing. Hyperdimensional spaces have a lot of capacity because of very low probability of collision between two representations.

An alternate representation of a word stored in a sparse distributed memory can be the listing of all the hard locations it is stored in. This is because if we read from the centroid of all the hard locations, we will obtain the word that was stored, and that this representation represents. If we use a binary vector for this listing of hard locations, where an on bit represents that the word is stored in that hard location and an off bit shows that it is not, then we obtain a sparse representation of this word. This is because a word is stored only in a very small percentage of hard locations. This sparse representation is very noise robust as sparse representations generally are (described in the following Section). Two such representations that are sufficiently similar will represent the same word stored in the SDM. This is because if two words have sufficiently similar vicinity of hard locations that they are stored in, then a read operation on this memory from these two similar vicinities will result in the same word being read.

6. Properties of Sparse Representations

The claim that sparse representation is useful for representing information is backed by the mathematics of its fault tolerance and noise robustness capabilities, along with its capacity to store information. It has been shown that sparse representations have massive representational power (Zhang, Xu, Yang, Li, & Zhang, 2015). They are very robust to noise, so much so that they can maintain reliable classification performance even with as much as 50% noise, as shown in the sub-section 6.4. The following description is based on a discussion of properties of sparse representations by Ahmad and Hawkins (2015).

6.1. Definitions

6.1.1. Sparse representations.

A sparse representation is an n-dimensional vector of binary components, like $x= [b_0, b_1, b_2, \ldots, b_{n-1}]$. These vectors are highly sparse, i.e. only a small percentage of their binary components are 1 and the rest are 0. We will use w_x to denote the number of components in x that are 1.

6.1.2. Overlap.

Overlap is a method of estimating similarity between two sparse representations. Overlap is more suited to measure similarity of sparse representations as compared to Hamming distance (number of positions at which the corresponding bits are different) because it helps in demonstrating some useful properties of these representations, discussed later. Similarity between two sparse representations is measured by an overlap score. The overlap score is the number of bits that are 1 in the same location in the two vectors. Overlap between two binary vectors, x and y, can be computed as the dot product of the two vectors;

$$overlap(x, y) = x.y$$

6.1.3. Overlap set.

The Overlap Set Ω(n, w, b) is the set of all vectors of size n and w active bits that have an overlap of exactly b bits with a given vector. The cardinality of this set is given by

$$|\Omega_x(n, w, b)| = C_b^{w_x}.C_{w-b}^{n-w_x}$$

where,

C_j^i is the number of possible combinations of j elements taken out of a total of i elements.

6.1.4. Matching.

We consider that two sparse representations represent essentially the same thing when they match. The criterion for matching is having an overlap score greater than a predefined threshold between the two sparse representation. This is defined as a match between the two sparse representations.

$$match(x, y)\ iff\ \ overlap(x, y) >= \theta$$

Typically, θ is set such that $\theta <= w_x$ and $\theta <= w_y$.

To illustrate these concepts with an example, consider two sparse vectors:

$$X = [00010000110000000100]$$

$$Y = [00010000110001000001]$$

Both vectors have dimensionality n=20, along with $w_x=4$ and $w_y=5$. The overlap score between them is 3. If the matching threshold θ is set at 3 then these two representations are considered to match. It may not be apparent why these two representations can be considered to match here, in the general sense of the word matching since they are not identical, but using the

properties of sparse representations that follow, we will show why it can be considered to match in the special case of high dimensional vectors. This is how two similar concepts can be distinct but also have similar representations.

6.2. Capacity

With this property we try to measure the number of unique sparse representations possible given an 'n' and 'w'. The capacity or the number of possible unique sparse representations can be given by

$$C_{\omega}^{n} = {}^{n!}/_{(n-w)!\,w!}$$

For typical values of these variables like n=2048 and w=40, the number of unique sparse representations is 2.37×10^{84}. This is an astronomically large number. The number of atoms in the observable universe are $\sim 10^{80}$. Therefore, we can safely claim that sparse representations have a very large capacity.

6.3. Uniqueness

The probability of two random sparse representations being identical is extremely small for all practical values of n and w. This is the uniqueness property of sparse representations. This probability is given by

$$P(x = y) \;=\; 1/C_{w}^{n}$$

For n=1024 and w=2 the probability is 1 in 523,776. This probability decreases steeply as w increases. For n=1024 and w=4 this becomes 1 in 45 billion. For the typical values, i.e. n=2048 and w=40 this number is essentially 0.

6.4. False Matching

False matching is where we can clearly see the noise robustness property of sparse representations. False matching is the probability of two random vectors having an overlap(x,y) score >= θ, so that they are falsely considered to be matching, even when they are random. This is given by

$$fp^{nw}(\theta) = \Sigma_{b=\theta}^{w} |\Omega_x(n,w,b)|/C_w^n$$

When θ is half of w then we assume that even with 50% of on bits different, i.e. 50% noise, we consider the two representations to match. So, for n=1024, w=4 and θ=2 (corresponds to 50% noise) the probability of a false match is 1 in 14,587, which is high. But if we increase w and θ to 20 and 10 respectively, this probability reduces drastically to less than 1 in 10^{13}. Thus, for any practical values of n, w and θ the probability of an error even with a very low threshold (50% of w) is extremely low. Therefore, we can tolerate up to 50% noise with practically perfect robustness. With θ<w/2 this probability of error enters the range where it no longer tolerable for practical use.

7. Methods of Producing Sparse Representations

I will categorize the methods based on the field of study that they derive from. There are three main fields: information theory and signal processing, machine learning, and theoretical

neuroscience. Like much of computer science, the oldest methods appear to come from the information theory and signal processing category. Machine learning methods are a newer branch of methods. Neuroscience based methods are the newest and, in my opinion, least recognized in the community.

It would not be possible to provide a comprehensive list of all methods of producing sparse representations here, but I will mention a few methods from each of these categories and a couple of methods in detail. I believe this will allow the reader to get a general understanding of methods and some detail into the methods relevant in this work.

In the field of signal processing, (Candes & Romberg, 2005; Donoho, 2006) proposed methods of compressed sensing and signal reconstruction form a sparse set of measurements. They focused on the limits on the number of measurements and sparsity for perfect reconstruction. They used L_1 minimization methods to obtain a sparse set of measurements, which corresponds to Fourier transforms of signals.

7.1. Methods in Machine Learning

Machine learning methods of producing sparse representations evolved in the domain of neural networks where the activity of a hidden layer of a three layer (input, hidden and output layers) neural network is the sparse encoding or representation of an input to the neural network. Autoencoders are the most popular form of producing sparse representations in the machine learning community. Like the compressed sensing methods, they try to reconstruct the input signal as best as possible. Different from regular autoencoders, sparse autoencoders have an

additional constraint on the encoding of an input that enforces sparsity (Lee, Battle, Raina, & Ng, 2007; Ng & others, 2011). The loss function of such a neural network has two main components, one that ensures a good reconstruction of the input signal at the output layer and a sparsity constraint on the encoding layer. Such a loss function is given below.

$$Loss = \Sigma_i \|x_i - y_i\|_2 + \beta \|a_i\|_1$$

where,

x is the input signal

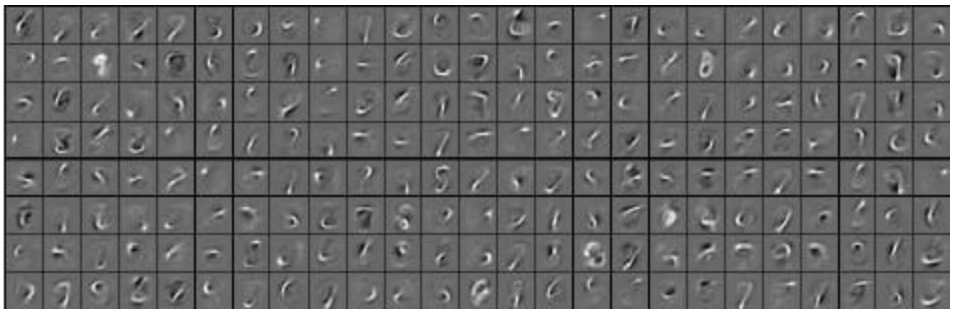
y is the output signal

i is the set of inputs

β is the weight of the sparsity term in loss function

a is the activation of the units of the encoding layer of the autoencoder

Such a neural network was demonstrated to learn pen stroke-like features, as shown in Figure 2, from the dataset of handwritten images MNIST (LeCun, Bottou, Bengio, Haffner, & others, 1998), by (Poultney, Chopra, Cun, & others, 2007; Steinert, Rehn, & Lansner, 2006). This shows that learning sparse features can enable learning of an explainable and interpretable set of features when the input domain has a structure where a few latent variables out of a multitude, produce a given sample.



(Ng & others, 2011)

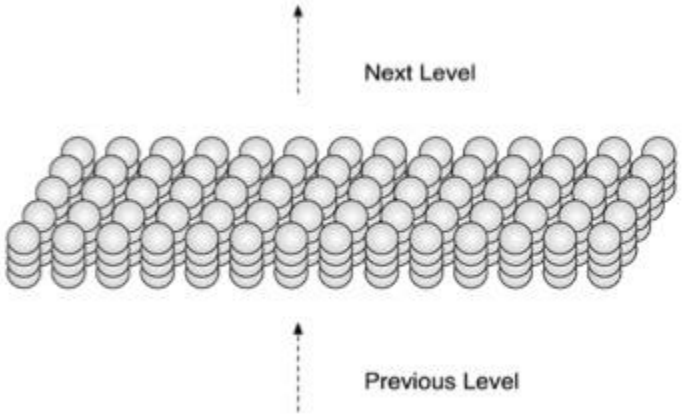
Figure 2 Features learned by sparse autoencoder when trained on the MNIST dataset of handwritten digits. Each box represents the feature learned by one hidden unit of the sparse autoencoder. The features resemble pen-strokes. Only a few pen strokes out of all the ones available can be used to create a digit.

7.2. Methods in Theoretical Neuroscience

Sparse representations have been studied in the context of theoretical neuroscience as a possible model by which the brain represents and stores information. Kanerva (1988) proposes a method of storing binary vectors in an auto-associative memory created out of sparsely placed hard locations where vectors are stored. An auto-associative memory can store data and can retrieve that data later by just a partial or a noisy version of this data. He proposes this as a candidate for the memory model of the brain, which has a lot of features similar to that of auto-associative memories. Olshaulsen and Field proposed the theory of sparse coding for brains (Olshausen & Field, 1997), as described in Section 3. There are a few other models inspired from neuroscience that try to describe the functional algorithm of the regions of the brain for the processing of information. Rinkus (Rinkus, 1995, 2004, 2014) describes the functional model of a microcolumn in the neocortex of the brain as a collection of units that act like neurons in the microcolumn. He enforces a sparsity constraint on this collection of units, based on various neuroscience-based evidence (Berger, Perin, Silberberg, & Markram, 2009; Kawaguchi & Shindou, 1998) about inhibitory neurons being able to perform such a function.

Hawkins et. al. (2005, 2009; 2011) have described another functional model of a region of the neocortex in the brain, which is very similar at the functional level to Rinkus' (Rinkus, 2014) model. Hawkins' model, called Hierarchical Temporal Memory (HTM), uses sparse representations extensively to represent patterns in an incoming stream of time series data. It uses a hierarchical stack of spatial and temporal pattern recognizers for processing these inputs. Sparse representations of temporal patterns are formed over sparse representations of spatial invariances in the input.

HTM uses a region of columns of cells to accomplish this, as shown in Figure 3. Each column has a natural receptive field in the input and obtains some activation proportional to the number of bits active in its receptive field. By a method of inhibition between columns it achieves a sparse set of columns that emerge winners after inhibition and are used as active columns for representing the given input. The cells in each column can be in either inactive, active or predictive states based on the activity of the cells nearby in the same column or other columns. A predictive cell indicates that the composing column is predicted to get activated in the next input. This is achieved by increasing and decreasing the weights of the connections, between cells of various columns, based on the correctness of the prediction. Following is a visual representation of cells and columns in HTM with their receptive fields.



Taken from https://www.taoeffect.com/other/nupic/

Figure 3 This shows an HTM Region. The columns are arranged in a 2D grid with each column having four cells.

8. Structure of this Dissertation

So far, I have discussed the motivation, uses, properties and methods of producing sparse representations. In the following two chapters I demonstrate applications of sparse representation in solving problems in AI and Molecular Biology. The second chapter is a paper I presented at the Sixth Advances in Cognitive Systems conference held at Stanford University in 2018. This paper describes using one of the methods of producing sparse representations in the sensory memory of a cognitive architecture. This paper shows how we can create a sensory memory that can produce rich grounded symbolic representations using Hierarchical Temporal Memory for the Learning Intelligent Decision Agent (LIDA) cognitive architecture. The third chapter is the draft of a paper that is to be submitted to a bioinformatics journal. This describes another method of producing sparse representations by sparse autoencoders to learn patterns in gene expression data. A sparse autoencoder was used in an unsupervised learning framework to represent a gene expression sample using a sparse set of features learned in the hidden layer of this neural

network. We demonstrate how biologically interpretable and relevant features were learned by this autoencoder. The fourth chapter presents my concluding remarks on sparse representations and its applications described in this dissertation.

CHAPTER 2: Sensory Memory for Grounded Representations in a Cognitive Architecture

By Pulin Agrawal, Javier Snaider, Stan Franklin

Abstract

Vector LIDA was proposed in 2014 as major overhaul to the representation system in the LIDA model. It replaced the nodes and links representation system used earlier with Modular Composite Representation vectors for representing knowledge in LIDA. Although, it was mostly trivial to swap out nodes and links with these vectors, as described in the Vector LIDA paper, it was not trivial to obtain these vectors from sensors so that symbol grounding is maintained. In this paper, we propose a sensory memory system and a method for obtaining Modular Composite Representation vectors from it in a way that enables symbol grounding. We propose to build sensory memory out of HTM Cortical Learning Algorithm regions, an online learning algorithm, which produce sparse distributed representations of spatio-temporal patterns recognized in its stimulus from the environment. These sparse distributed representations can then be translated into Modular Composite Representation vectors that preserve the semantic meanings encoded in the sparse distributed representations, and thus enable the LIDA's vector representations to be grounded by the sensors and the environment. One of the main commitments of the LIDA model is that "all learning happens via consciousness". To allow HTM Cortical Learning Algorithms to learn and adapt to the environment and still honor this commitment, we need to translate the contents of conscious broadcast to sparse distributed representations, so that Cortical Learning Algorithms can learn based on consciousness. To summarize, we propose a Cortical Learning Algorithms based Sensory Memory for the Vector LIDA Model and produce a method of translating back and forth between the representations used by Cortical Learning Algoritms, i.e.

sparse distributed representations, and the representations used by Vector LIDA, i.e. Modular Composite Representation vectors. This will allow symbol grounded Vector LIDA based agents to be developed.

1. Introduction

"Once a problem is described using an appropriate representation, the problem is almost solved." (Winston, 1992, p. 18). Many authors have stressed the importance of well-chosen representations in trying to solve a problem or perform a task. For a cognitive architecture, the task of choosing the right representation system becomes crucial because its agents may be expected to represent a variety of complex forms of information, and to perform an equally varied set of tasks. The importance of representation for symbolic and connectionist cognitive architectures has been discussed in more detail in (Franklin, 1995, p. 365). Ron Sun (1999) talks about the need to integrate the two most popular forms of representations, symbolic and connectionist, to reap the benefits of both.

Many cognitive architectures choose to use a symbolic representation system because that provides a robust way to manipulate the knowledge represented by an agent, and to think and talk about it for people observing the agent or interacting with it. We believe that these symbolic representations need to be grounded in the environment of the agents that are built using these cognitive architectures. If we want an agent to be able to perform well in a highly complex and dynamic environment, it is important to have symbol grounding in order to capture the dynamism of the environment. This can only be done when the representations are grounded in environment. Symbol grounding is achieved by the sensory memory of a cognitive architecture. If the representations that come out of the sensory memory faithfully deliver the required

complexity and dynamism from the environment to the rest of the system, we can say that the representation system of the cognitive architecture is grounded in the environment.

But it is not enough to just produce grounded representations that faithfully represent the environment. A cognitive architecture also needs to make sure that the processes and memory systems it employs are able to exploit all the needed nuances that are captured by these representations. For example, the recognition memory system must be able to recognize learned objects/concepts from representations of what the agent has sensed in the environment. Sensory memory and the recognition memory should be able to communicate seamlessly and with high fidelity so as to provide symbols in the recognition memory the expressive power of the representation to be used by other parts of the system.

Here we provide a solution to the two problems of trying to create a sort of hybrid representational system that has the benefits of both symbolic and connectionist representations, and building a sensory memory that produces grounded representations that can be communicated seamlessly and with high fidelity to the rest of the cognitive architecture. We solve these problems in the context of a systems-level cognitive architecture called LIDA (Franklin et al., 2016), short for Learning Intelligent Decision Agent.

Recent advancements to the representation system in LIDA model, by the virtue of using Modular Composite Representations as a part of Vector LIDA project (Snaider & Franklin, 2014b), makes LIDA a solid contender among cognitive architectures. The value of using the new representation system for LIDA was argued for in (Snaider & Franklin, 2014b). This new

representational system makes it much more powerful by preserving all the benefits of its earlier symbolic representations in the form of nodes and links, and adding processing capabilities that were not possible with nodes and links. In addition, this system makes the representations very noise robust. We feel that the earlier sensory memory system does not do justice to the augmented capabilities of LIDA model with this new representation system. The earlier sensory memory still uses nodes and links, which are not expressive enough for information rich dynamic environments that Vector LIDA is capable of representing with the help of Modular Composite Representation vectors. In addition to that, the only way of creating Modular Composite Representation vectors for an activated node in Sensory Memory is to create a random integer vector for that feature. This makes the symbol grounding weak in LIDA model.

In this paper we propose a new Sensory Memory System for LIDA that compliments the new vector based representational system of Vector LIDA by delivering richer representations than nodes and links. Sensory representations now will be in the form of high dimensional Sparse Distributed Representations, which have greater expressive power and thus enables stronger symbol grounding. This creates a need for translation between the two types of representations, viz. Modular Composite Representation and sparse distributed representation, for the sensory memory and recognition memory to be able to communicate. In this paper we also describe a method of doing high fidelity translation between sparse distributed representation and Modular Composite Representation vectors that preserves the richness of information during translation. We describe the two major contributions mentioned above in the Sections 3 (HTM as Sensory Memory) and 4 (Translation Problem) respectively. Section 2 will give the reader some background about the concepts that we talk about in the rest of the paper. Section 5 describes the

experiments done to validate the claim of high-fidelity seamless translation between Modular Composite Representation and sparse distributed representation vectors. Section 6 will conclude with a discussion of the results and contributions of this paper.

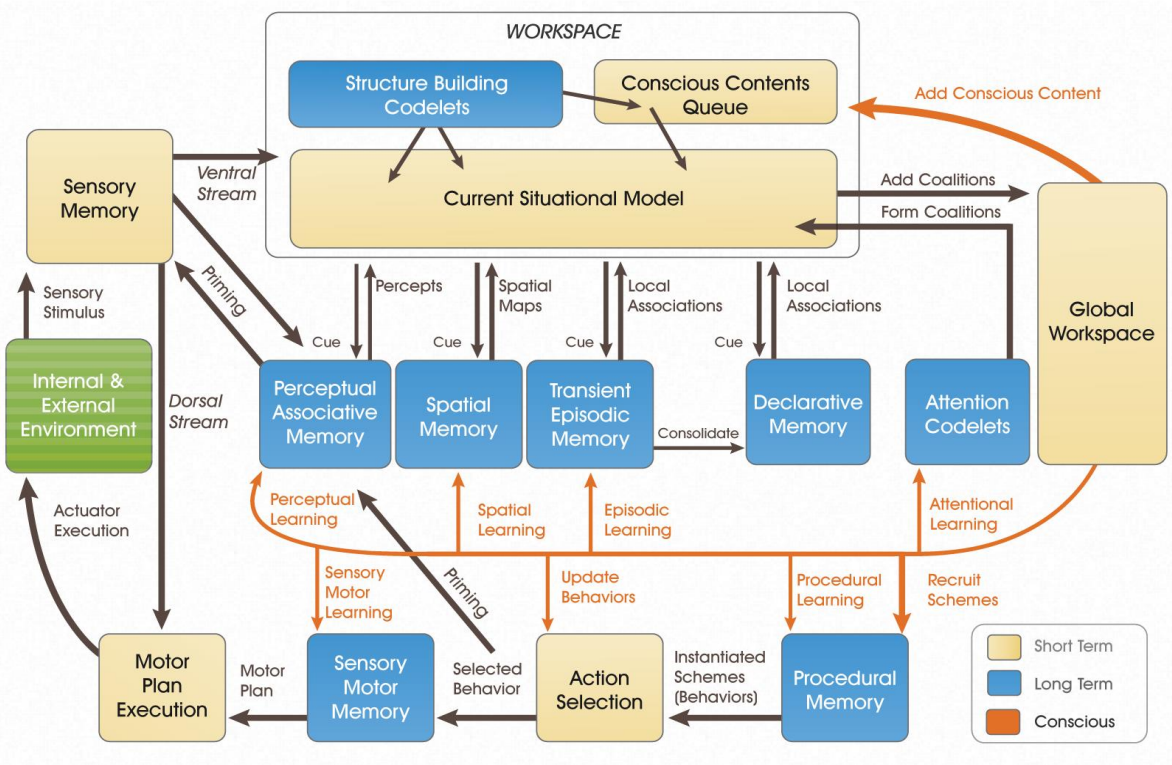
2. Background

2.1. LIDA Model and GWT

Cognition is said to be made up perception-action cycles by neuroscientists and psychologists (Cutsuridis, Hussain, & Taylor, 2011; Dijkstra, Schöner, & Gielen, 1994; Freeman, 2002; Fuster, 2004; Fuster, 2002; Neisser, 1976). An agent (natural or artificial) samples its environment, internal or external, to initiate the process of perception; with the rest of the system employing other cognitive processes it completes the cycle by performing an action. This is what a cognitive cycle is. One important cognitive process in this cycle is attention. Attention filters for the most salient aspects of the current situation for the agent, and broadcasts them to the whole cognitive system to use as conscious content in accordance with Global Workspace Theory (Baars, 1988).

The LIDA model (Figure 4) is built out of these cognitive atoms, aka the cognitive cycles, running in parallel in an agent. Various processes of cognition, like planning, imagining and reasoning, happen over multiple cognitive cycles. These processes are enabled by various modules working together. They work by manipulating and restructuring data during a cognitive cycle working towards the fulfillment of the agenda of the agent. For example, attention codelets produce coalitions, including salient content from aspects of the current situation, that compete for consciousness. Similarly, structure building codelets build new structures from the contents of current situation. Each cognitive cycle is marked by the event of conscious content broadcast, which initiates learning in possibly all the various memories of the LIDA model. This will be

relevant later in this paper, because this is one of the main conceptual commitments of LIDA model to which we must adhere in the process of building a Sensory Memory. The commitment comes from the Conscious Learning Hypothesis (Franklin, Strain, McCall, & Baars, 2013), and states that significant learning takes place via the interaction of consciousness with the various memory systems. This hypothesis has been argued for in detail in (Franklin, et al., 2013) Consciousness refers to functional consciousness here, as opposed to phenomenal consciousness. Consciousness acts like an attention filter in LIDA.



Taken from https://ccrg.cs.memphis.edu

Figure 4 The LIDA Cognitive Model.

Further details about the LIDA model are beyond the scope of this paper, and can be found in the LIDA Model Tutorial (Franklin, et al., 2016). In the next two sections we discuss the two important memories in LIDA Model that are relevant for discussions in this paper.

2.1.1. Sensory memory.

Sensory Memory in LIDA uses feature detectors. Feature detectors are daemons which are constantly looking for incidences of their interest in the environment. One feature detector is tasked with looking at one particular feature in the entire perceptual scene. When it detects that feature in the environment it puts a node in Sensory Memory that lasts for a few milliseconds. This node represents the feature and tells the agent that that particular feature was recently present in the environment. Since the nodes in the Sensory Memory last only a few milliseconds, it is a short-term memory.

Sensory Memory plays an important role in fulfilling the Embodied Cognition commitments of the LIDA model (Franklin, et al., 2013). Embodied Cognition claims that the body as well as the brain (if it has one) of an agent is situated in an environment, and the relationship between environment, body and mind affects all cognitive processing (de Vega, Glenberg, & Graesser, 2008). The LIDA model adheres to Embodied Cognition by using only perceptual symbols (Barsalou, 1999). Sensory memory is directly involved in creating these perceptual symbols that faithfully represent the environment of the agent.

2.1.2. Perceptual associative memory.

Perceptual Associative Memory (PAM) is the recognition memory of LIDA. It uses nodes and links to represent information. Nodes represent features, objects, feelings, actions, events, categories etc. Links represent relations between these nodes, like feature-of, causation, category membership etc (See Figure 5). These are the things an agent has learned to recognize in its

environment, and their relevant relationships to one another. Structures in PAM can be accessed by cues from the Current Situational Model (See Figure 4). They are activated from other PAM nodes or from Sensory Memory. This allows the agent to form percepts from the sensory stimulus that are then sent back to Current Situational Model to enable the agent to have a better understanding of the current situation that it is in with regard to its internal and external environment.

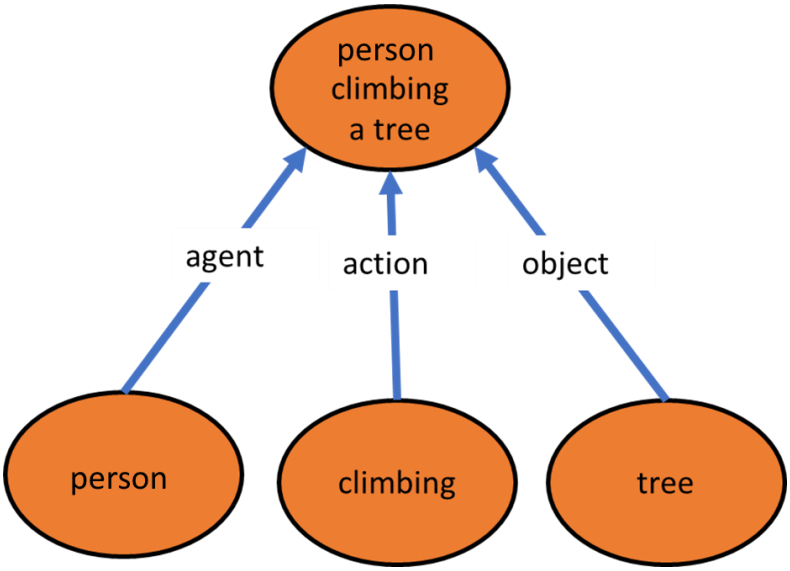


Figure 5 A node and links structure in PAM representing an event.

2.2. Vector LIDA

Vector LIDA was presented as a major overhaul of the representation system of the LIDA model. It uses high dimensional Modular Composite Representation vectors, instead of nodes and links as previously described. These Modular Composite Representation vectors are stored in Integer Sparse Distributed Memory (Snaider, Franklin, Strain, & George, 2013), which is a modified version of Kanerva's (1988) Sparse Distributed Memory, for storing integer vectors instead of binary vectors (1988). There are several advantages to using Modular Composite Representation vectors for representation, and ISDM for memory. It, enhances the learning

mechanisms, improves scalability and provides better integration with episodic memory and sensory memory. For more details on that and other benefits please refer to the Vector LIDA paper (Snaider & Franklin, 2014b).

Since, Vector LIDA is based on representing information using Modular Composite Representation vectors and processing these vectors, we briefly describe Modular Composite Representation vectors in the following section.

2.2.1. Modular composite representation vectors

Modular Composite Representation (MCR) (Snaider & Franklin, 2014a) vectors are long integer vectors. They are similar to Spatter Code (Kanerva, 1994) in its properties, where Spatter Code uses binary vectors. Modular Composite Representation is a reduced description model. This means that complex representations can be compressed/reduced into one vector, and can be reconstructed out of this vector when needed. The interesting thing to note here is that each of the components of the complex representation is itself a long vector of same dimension as the compressed vector. The following paragraphs provide a little more technical description of Modular Composite Representation vectors.

A system using Modular Composite Representation vectors defines the size of the vectors it is going to use, which remains constant for that system's lifetime. Every Modular Composite Representation vector has that many dimensions, where each dimension can take a value from a defined range of integer values, for example, (0, 1, …, 15) (Aggleton & Pearce, 2001). More formally, a Modular Composite Representation vector is a vector in a multidimensional space,

$v \in \mathbb{Z}_r^n$, where n is the number of dimensions of the space and r denotes the range of integer values for each dimension.

Two basic operations defined on these vectors are Binding and Grouping. These are important concepts to understand for getting a grasp on the translation between sparse distributed representation and Modular Composite Representation vectors.

- Binding is like a multiplication operation, symbolized by $\otimes$. It is defined as the modular sum in each dimension. It works like the bitwise XOR for binary vectors. For example, multiplying vectors $X, Y \in \mathbb{Z}_{16}^n$ to get integer vector Z = X$\otimes$Y, if the dimension *i* in X and Y has values 11 and 14 respectively, then in the resulting dimension in Z, Z_i will be 9. Formally,

$$Z_i = mod_{16}(X_i + Y_i)$$

The unbinding operation can thus be defined as the modular subtraction in each dimension between the operands or as the binding of the first operand to the complement of the second operand X = Z$\otimes$ Y^{-1}. The complement, or the inverse vector, in this model is such that each dimension in Y^{-1} is the complement of that value in the same dimension. Continuing our previous example each dimension in Y^{-1} would be,

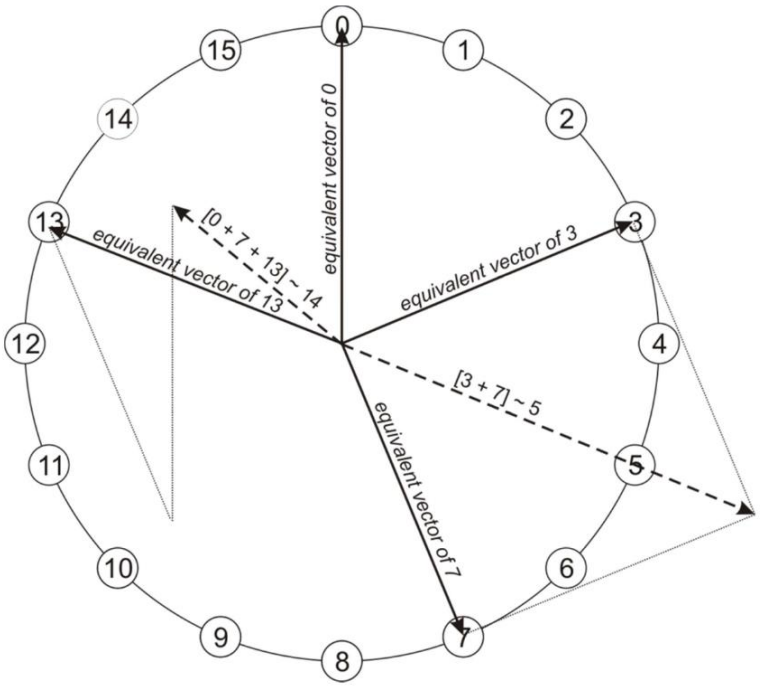
$$Y_i^{-1} = mod_{16}(16 - Y_i) = 16 - Y_i$$

Two important properties of this operation are that

- It is associative, commutative and distributive over the sum or grouping operation.
- It produces a vector that is very different from the operands. Almost always orthogonal.

- The grouping operation is like finding a resultant vector of two vectors, symbolized by +. To describe more formally, for each dimension we place the values of that dimension for each operand as an equivalent vector on a circle as shown in Figure 6. For example, the equivalent vector for the value 0 is (0,1), 4 is (1,0), 8 is (0,-1), 12 is (-1,0). The sum operation for each dimension will be the resultant vector of the equivalent vectors of all operands for that dimension after normalization and rounding to closest equivalent vector corresponding to an integer. For example, in Figure 6, the normalized result of equivalent vectors for 3 and 7 is 5. Another example in Figure 6 demonstrates rounding. When we add the three equivalent vectors 0, 7 and 13 the result is 14 after rounding.

  The most important property of the grouping operation is that the resulting vector is similar to each of its operand vectors.



(Snaider & Franklin, 2014a)

Figure 6 Finding equivalent vector for a value of a dimension.

These properties of the binding and grouping operations allow Modular Composite Representation vectors to act as reduced descriptions. A reduced description can be created from a collection of attributes and their names. For example, from two attributes Shape and Color, we can create a reduced description of a "red circle ". If we have vectors representing Shape, Color, red and circle, this can be done by using binding and grouping operations as follows:

$$F = [circle \otimes Shape + red \otimes Color]$$

This is a useful reduced description only if we can recover the components that created F. This can be done if we use an Integer Sparse Distributed Memory to store these vectors. We will

continue this example to show how to recover vectors from a reduced description using an ISDM, after we introduce ISDM in the next section.

2.2.2. Integer Sparse Distributed Memory

Integer Sparse Distributed Memory (ISDM) is like Kanerva's Sparse Distributed Memory (SDM), but it is used to store high dimensional integer vectors, for example Modular Composite Representation vectors. Like SDM it is an auto-associative memory. For details on the functioning of ISDM please refer to (Snaider, 2012). ISDM consists of a fixed number of hard-locations where vectors can be stored. A hard location is a point in the multi-dimensional space representing an integer vector, a point of the kind that this memory will eventually store. These hard locations are distributed uniformly randomly over the vector space. A vector is written in all the hard-locations that are within a critical distance from that vector. Since a vector is written in multiple locations it is noise robust. Also, when we try to read a previously written vector from the memory we do not need the exact vector to effect the read. A noisy version of the vector can be given to ISDM, which works like a cleanup memory. Since the noisy vector is like the vector we want to read, ISDM will look in approximately the same vicinity as it used to store the non-noisy vector, to find hard-locations to read this vector from. ISDM performs a kind of polling from all the hard locations that it finds within the critical distance to produce the vector that was originally written. The resulting vector is less noisy because of the polling. ISDM can then repeat this process until convergence to obtain the cleaned up or non-noisy vector. This is how we can read the original vector using a noisy version of that vector.

2.2.3. Reduced Description

In section 2.2.1 we created a reduced description of a red circle using red, circle, Shape and Color vectors as:

$$F = [circle \otimes Shape + red \otimes Color]$$

A reduced description has a role filler type structure, where Color and Shape are the roles and red and circle are corresponding fillers. For F to be a useful reduced description of "red circle" we need to be able to retrieve the attributes of the "red circle" from the vector F. We can do this using the method of probing.

2.2.3.1. Probing

Probing is the processes of checking to see if a reduced description vector contains the probed vector as a component.

For reduced descriptions created by the binding operation, this is done by binding the reduced description with the inverse of the role vector (unbinding). What we get from this process is the filler and noise. This noise can be removed by using cleanup memory to retrieve the actual filler, if we have all the vectors stored in an ISDM. For example, if we want to check the shape of the "red circle" we could probe its reduced description with the inverse of Shape.

$$F \otimes Shape^{-1} = [\, circle \otimes Shape \otimes Shape^{-1} + red \otimes Color \otimes Shape^{-1}]$$

Color, $Shape^{-1}$ and red, when bound, produce noise, and Shape and $Shape^{-1}$ when bound cancel out, so we get:

$$F \otimes Shape^{-1} = [\, circle + noise \,]$$

This noise can be removed by reading this resulting vector from a cleanup memory, so we come full circle. If F is probed with a role that was not a component to begin with, then all we get is noise, which results in an unsuccessful read operation from the cleanup memory.

For reduced descriptions created by grouping operations, probing is done by checking the distance[1] of the probed vector from the group vector. If the distance between a group vector and the probe vector is less than a threshold, we can say that the group vector was grouped/composed using the probe vector. This is possible because the group vector is similar to all of its component vectors. Thus, the distance between the group vector and any of its components is less than the distance between any two random vectors. Formally, for a vector G with components A and B,

$$G = [A + B]$$

$$distance(G, A) < Threshold$$

$$distance(G, B) < Threshold$$

$$distance(G, X) \geq Threshold$$

$$where, X\ is\ any\ random\ MCR\ vector$$

[1] Distance between two Modular Composite Representation vectors is defined as an extended Manhattan metric which accounts for the modularity of dimensions.

(Snaider, 2012) This threshold is the Indifference Distance of this Modular Composite Representation space, where Indifference Distance is defined as the distance between any two random vectors in this Modular Composite Representation space.

2.3. Hierarchical Temporal Memory

Hierarchical Temporal Memory is a spatio-temporal pattern recognizer capable of online learning, proposed by Hawkins et. al. (Hawkins, Ahmad, & Dubinsky, 2011). It is inspired by the computations in the layers of the neocortex in the brain (Hawkins, George, & Niemasik, 2009). It uses activity of a region of columns to represent a given input. It employs sparse distributed representations for representing input. This means that for any given input only a few of the columns, out of thousands, in a region are active, typically 2% to 5%. Sparse representations provide a lot of benefits for representation of a domain of inputs. They are noise robust. They also allow a lot of freedom and flexibility with which to encode similarity between inputs. Thus, it is very expressive. For more details please refer to (Hawkins, et al., 2011).

2.4. Problem Elaboration

We need to build a Sensory Memory for LIDA that will allow us to leverage the greater expressive power of the Modular Composite Representation vectors used in the Vector LIDA system, to replace the earlier simpler feature detectors used so far in the Sensory Memory. We intend to build the new Sensory Memory out of HTM Regions. Each HTM Region will be responsible for sensing an aspect of the environment. For example, one region can be used to sense the location of the agent from a GPS sensor, another region can be used to sense the motor

commands of the gripper of an arm of the agent. All the benefits of using HTM Regions for this task are explained in the next section.

We face a few challenges in building the Sensory Memory out of HTM Regions. The first challenge is that the representations coming out of HTM Regions are Sparse Distributed Representations, whereas the rest of the system in Vector LIDA is going to use Modular Composite Representation vectors. Therefore, we need to be able to translate sparse distributed representation vectors into Modular Composite Representation vectors. We need to do this in such a way that we do not violate the embodied cognition commitment of the LIDA model. We can do this only by preserving the meaning and expressiveness of the sparse distributed representation during translation, so that rest of the system can understand its environment effectively. Thus, the translation will not violate embodied cognition.

The second challenge is that we need to allow HTM Regions to employ online reinforcement learning. This is a challenge because if we want to allow HTM to learn during the lifetime of an agent we need to adhere to the Conscious Learning commitment of LIDA model, as mentioned in Section 2.1. To allow conscious learning we will only let HTM learn from the contents of the conscious broadcast. In Vector LIDA the conscious broadcast will consist of one or more Modular Composite Representation vectors. We need to be able to retrieve the lowest level sensory content from these vectors, and further translate them into sparse distributed representations, so that Sensory Memory Module can map these sparse distributed representations to the respective regions and columns, and perform reinforcement learning accordingly.

3. HTM as Sensory Memory

We want to build a Sensory Memory that can deliver information rich representations to PAM so that Modular Composite Representation based systems of Vector LIDA can exploit Modular Composite Representation's power of representation, and thus allow the agent to better deal with complex and dynamic environments.

LIDA's Sensory Memory can use HTM Regions as feature detectors. We can envision an agent having multiple HTM Regions to sense a variety of aspects of its environment. Even though HTM is not the state of the art in image processing and computer vision, as it improves it can be used to directly sense the environment through a camera. It has been shown to work for a video game environment on ray cast image data (Sungur, 2017). For now, it can still be used for simpler agents by doing edge detection and image segmentation in a pre-processing step, and feeding in the segmented filtered image as input to a HTM Region to detect shapes. Another HTM Region can be employed to detect colors in the segmented filtered image. Benefits of doing this include that similar shapes and similar colors will have similar representations created by HTM Regions. A big advantage of using HTM based regions is that they are agnostic of the input modality. We do not need to hand-craft the feature detectors for every feature and every modality. Unlike many other neural network models, they employ online learning. Depending on the need of the agent, HTM can be setup to extract as much granular information from the input as needed. For example, it can be setup so that it can detect Red and Pink to be similar colors, or very different, as needed by the agent.

The current implementation of HTM employs Encoders to feed data into the HTM. Encoders are a preprocessing layer for the input. Numenta has created Encoders for various type of data like Scalar, Category, Date, Coordinate, Geospatial Coordinate etc[2]. An agent needing to understand its acceleration (a three tuple Coordinate) from its accelerometer sensor, or its location (a Geospatial Coordinate) from the GPS sensor, can use these encoders to sense from its environment. Here too we can reap the benefits of similarity in input from the HTM's pattern recognition. An acceleration of similar magnitude or direction will have a similar representation by HTM. For example, an acceleration due North will have similar representation to an acceleration due Northwest, but completely different representation to one due West. A location near to another location will have a similar representation, and these are tunable by the parameters of these Encoders to suit the needs of the agent. In this way we can have HTM sense a variety of attributes of the environment for an agent with a high degree of control on the expressiveness.

It is important to note that the LIDA model is a conceptual framework, and that agents are specific implementations crafted from this framework, utilizing some or all the concepts available in the framework. An agent can use many different HTM Regions based on its need, and the HTM Regions will be configured so that it can deliver representations that are relevantly information rich for its purposes.

The proposed HTM based Sensory Memory allows for flexibility between how much pre-learned structures are needed and how much can be learned. If we need some pre-learned

[2] https://github.com/numenta/nupic/tree/master/src/nupic/encoders

structures, we can train an HTM Region before using it for the agent in the same, a simulated or a similar environment. If we need it to learn everything from scratch while living in the environment, we can use untrained HTM Regions.

4. Translation Problem

Sparse Distributed Representations produced by HTM Regions and Modular Composite Representations used for representations in LIDA are both high dimensional vectors but are very different in nature. sparse distributed representations are sparse binary vectors, in which only a few dimensions have 1s and rest are 0s. Modular Composite Representations are integer modular vectors, in which each dimension can have an integer value between a specified range. HTM Regions will represent a stimulus in the form of sparse distributed representations. PAM needs to use these sparse distributed representations to start recognizing relevant objects/percepts in its environment. Since PAM is built out of Modular Composite Representation vectors, we need to create Modular Composite Representation vectors out of sparse distributed representations produced by HTM Regions, so that PAM can use them meaningfully for recognition. This means that similar sparse distributed representations should produce Modular Composite Representation vectors that are close in Modular Composite Representation vector space. Thus, their similarity, and therefore meaning, in Modular Composite Representation vector space is preserved.

We need to create these Modular Composite Representation vectors in such way that we can get the sparse distributed representation vectors back from the Modular Composite Representation vectors. We need to do this because of the conscious learning commitment of LIDA model. Since, the HTM Region employs online learning, if we want the HTM Regions in

Sensory Memory to learn during the lifetime of the agent, we need to adhere to the conscious learning commitment, as also mentioned in Section 2.4, and provide only the contents of conscious broadcast for it use for learning. The contents of conscious broadcast from a Vector LIDA based agent will be in the form of one or more Modular Composite Representation vectors. We need to retrieve the sensory content (in the form of sparse distributed representations) from these Modular Composite Representation vectors, and give it to the respective HTM Region from whence it came, for the HTM Region to be able to learn from it.

Thus, the translation problem is a two-part problem of translating sparse distributed representation vectors to Modular Composite Representation vectors in such a way that they preserve their meaning upon translation, and they can be later retrieved back for learning in HTM Regions, if needed. We first describe the solution to translating a sparse distributed representation vector to an Modular Composite Representation vector in the next Section 4.1. In the following Section 4.2, we describe how to get sparse distributed representations from Modular Composite Representation vector based conscious contents.

4.1. Sparse Distributed Representation to Modular Composite Representation

The inspiration for the solution of the problem of trying to convert sparse distributed representation vectors to Modular Composite Representation vectors comes from a technique in mathematics called Random Projection (Bingham & Mannila, 2001). Random projection gives us a method of projecting a high-dimensional vector space to a lower dimensional vector space, in a way which approximately preserves the distances between the points, as determined by the Johnson-Lindenstrauss lemma (Dasgupta & Gupta, 1999). In random projection, the original d-

dimensional data is projected to a k-dimensional (k << d) subspace through the origin, using a random $k \times d$ matrix R, where every column is orthogonal to all other columns. If the original data of N points is in matrix $X_{d \times N}$. Then we get the projection $P_{k \times N}$ as:

$$P_{k \times N} = R_{k \times d} \, . \, X_{d \times N}$$

Here, we store our sparse distributed representations in the matrix $X_{d \times N}$, where d is the dimensionality of the sparse distributed representation and N is the number of sparse distributed representations we want to translate. We generate the random $R_{k \times d}$ matrix, per HTM Region, by generating d random Modular Composite Representation vectors, of the size defined to be used in the agent, here k. Since, it is a property of random Modular Composite Representation vectors that they are orthogonal to each other, we satisfy the requirement of generating an appropriate R. Thus, by multiplying the matrices and using the modular addition within a dimension as described in the grouping operation, we can get the resulting projected data in Modular Composite Representation vector space in $P_{k \times N}$, in the form of N k-dimensional Modular Composite Representation vectors.

Abstractly, we can describe this process of translation as treating each dimension of a sparse distributed representation vector as a feature, and generating a random Modular Composite Representation vector for each feature. This random Modular Composite Representation vector represents its feature in Modular Composite Representation space. Now, whenever we get a sparse distributed representation with some features active, we generate a Modular Composite Representation vector with the grouping of all the Modular Composite Representation vectors

that represent all the active features, as shown in Figure 7. For a sparse distributed representation vector X we generate a projected vector Y using M, where M is a set of randomly generated Modular Composite Representation vectors one for each dimension in X indexed by i as M^i, as:

$$Y = \sum_{i \in \boldsymbol{I}} M^i,$$

where, $\boldsymbol{I}$ is the set of indices of dimensions in X with an active bit and Σ is the grouping operation.

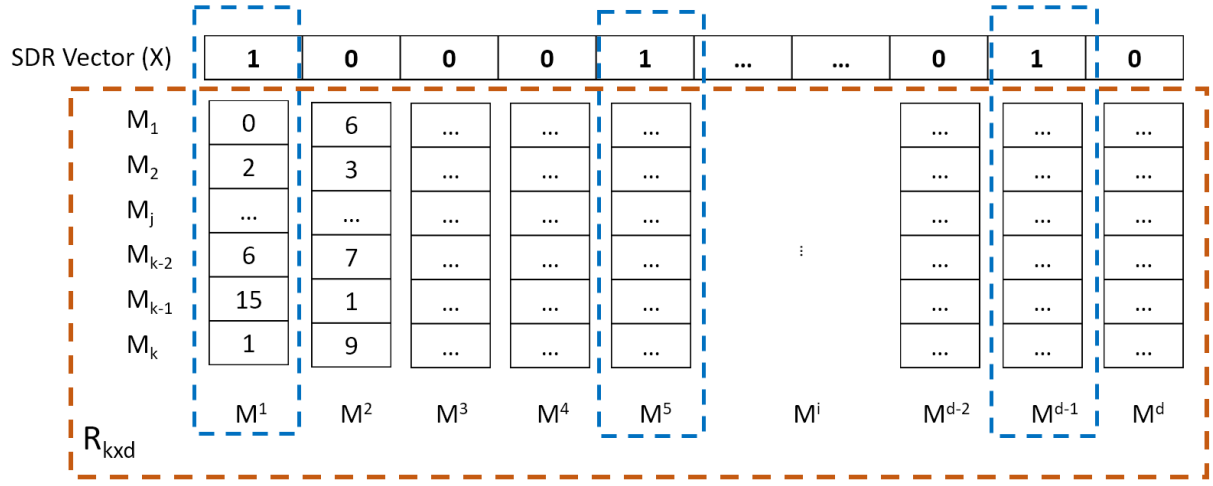


Figure 7 Projected vector of X is Y. Where Y is the sum (as defined by grouping operation) of all the Modular Composite Representation vectors M^i where sparse distributed representation vector X_i has a 1, as shown by blue boxes. The orange box shows the abstract random projection matrix R_{kxd}.

We need to be able to distinguish between Modular Composite Representation vectors generated by sparse distributed representations from two different HTM Regions co-existing in an agent. For this reason, we generate a random Modular Composite Representation vector and assign it to each region, as a Modular Composite Representation vector representing that region. Once we translate a sparse distributed representation we can bind (using the binding operation defined in Section 2.2.1) the resulting Modular Composite Representation vector with the

Modular Composite Representation vector assigned to the region from which the sparse distributed representation came from. Now, the resulting vector is ready to be used by PAM. Optionally, we can also additionally bind it to a Modular Composite Representation vector that represents the sensory modality which the HTM Region is looking at. For example, we may have multiple HTM Regions employed for each of the various modalities of the agent like visual, auditory, motor, internal environment etc.

4.2. Modular Composite Representation to Sparse Distributed Representation

A LIDA based agent will get a Modular Composite Representation vector or a group of Modular Composite Representation vectors from each conscious broadcast. A Modular Composite Representation vector will be a reduced description of some kind. We can use the unbinding probing operation to determine if the vector has any content from any of the HTM Regions in that agent. We retrieve the filler from any successful probes with any HTM Regions. Once we get the filler for a HTM Region we can then use the grouping probe operation using all M^i vectors as probes one by one. The M^i vectors that pass the probing indicate that the corresponding feature of the sparse distributed representation dimension was active. This way we can reconstruct the sparse distributed representation from the Modular Composite Representation vector. Once we reconstruct X, it can be used by the corresponding HTM Region for learning. Formally, to reconstruct each dimension i of X we probe the Modular Composite Representation vector Y with the corresponding Modular Composite Representation vector M^i.

$$X_i = \begin{cases} 1, & distance(Y, M^i) < Threshold \\ 0, & distance(Y, M^i) \geq Threshold \end{cases}$$

We use a stricter threshold of grouping probe operation than used in (Snaider, 2012). Here we use μ_{id}-k*σ_{id} as the Threshold, where μ_{id} and σ_{id} are the mean and standard deviation of the Indifference Distance between the Modular Composite Representation vectors of the specified length, and k is a parameter of the method that can be varied to optimize the performance of reconstruction.

5. Experimental Studies

To establish the validity of our method of translation we need to prove two things. First, this method must preserve the distance between two vectors in translation, i.e. different sparse distributed representation vectors when projected should be orthogonal Modular Composite Representation vectors, and similar sparse distributed representation vectors, when projected, should result in Modular Composite Representation vectors that are very close in Modular Composite Representation space. Second, the reconstruction from a Modular Composite Representation vector back to an sparse distributed representation vector is of high fidelity, and preserves most of the information. We can tolerate some information loss in the translation back to sparse distributed representation vectors because sparse distributed representation vectors are very noise robust.

For the first test we generate 100 random sparse distributed representation vectors, 1024 dimensions long and 2% sparse, this is a typical dimensionality for sparse distributed representation vectors from HTM Regions. We then translate them, as described in Section 4.1, into corresponding Modular Composite Representation vectors of dimension 1024, also typical for Modular Composite Representation vectors. We call this set of Modular Composite Representation vectors a reference set. To obtain similar sparse distributed representation vectors and check the quality of translation we add some noise to the sparse distributed representation

vectors. Less noise means that sparse distributed representation vectors are similar more noise means they are different to the point where 100% noise means completely different sparse distributed representation vectors. So, we add *x*% noise to the sparse distributed representation vectors by randomly changing *x*% of active bits to inactive and activating the same number of other randomly chosen inactive bits, thus maintaining the sparsity at the same level. In other words, adding x% noise to a 1024-dimensional sparse distributed representation vector creates a new sparse distributed representation vector that has a Hamming distance of $(2 * x/100 * 1024)$ from the original vector. We then use these noisy sparse distributed representation vectors to generate a new set of Modular Composite Representation vectors. We measure the distance of Modular Composite Representation vectors in the reference set to the ones generated by corresponding noisy sparse distributed representations. Figure 8 graphs the results of this measurement for the 100 vectors.

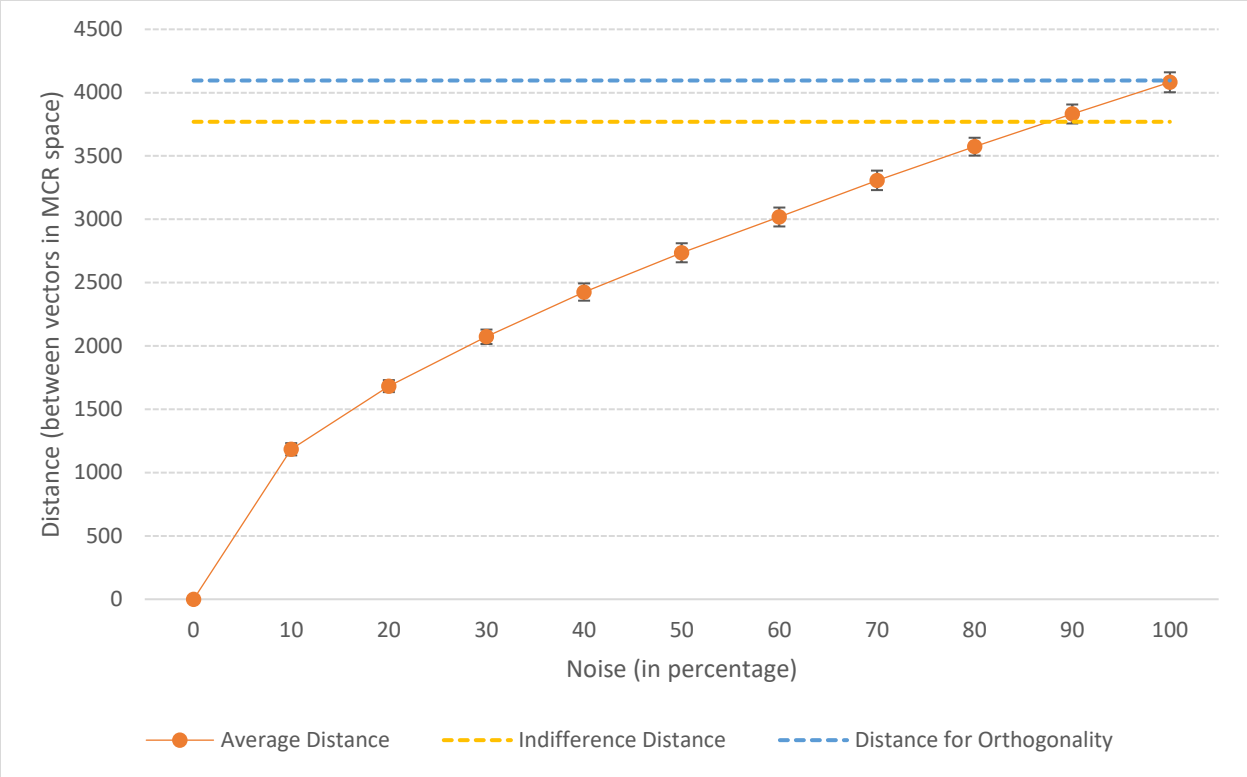


Figure 8 This graph shows the average distance between translated Modular Composite Representation vectors, before and after adding noise to the corresponding sparse distributed representation vectors. Standard deviation bars are shown on each point at the measured noise level.

For the second test we take the reference set of the 100 Modular Composite Representation vectors and translate them back to sparse distributed representation vectors using the method described in Section 4.2. We then compute precision and recall for the number of dimensions of a sparse distributed representation correctly retrieved for each of the Modular Composite Representation vectors. The statistics of the precision and recall for 100 vectors for different values of the parameter k are recorded in the table in Table 1:

Table 1 This table shows the statistics of precision and recall of retrieval of dimension values of sparse distributed representation vectors from Modular Composite Representation vectors for various value of the parameter k.

k	0	1	2	3	4	5
Precision	0.038 (±0.001)	0.110 (±0.007)	0.461 (±0.052)	0.943 (±0.050)	0.999 (±0.006)	1.0 (±0.0)
Recall	1.0 (±0.0)	1.0 (±0.0)	1.0 (±0.0)	1.0 (±0.0)	1.0 (±0.0)	1.0 (±0.0)

We can see from Table 6, that this system is able to reconstruct the sparse distributed representations with perfect precision and recall for k=5. We have made the code that produced these results publicly available[3].

6. Discussion

We needed the representations in the Vector LIDA model to be grounded and, we wanted to build a Sensory Memory that could do justice to the power of Vector LIDA by delivering an information rich representation of the environment to the model. We demonstrated how HTM Regions can be used to build high quality feature detectors for Vector LIDA model. These HTM Regions can be configured to be as expressive in capturing the information from the environment as needed by the particular agent. We also demonstrated how the HTM based Sensory Memory can facilitate successful symbol grounding by achieving high quality translation from HTM representations (sparse distributed representations) and Vector LIDA representations (Modular Composite Representation vectors) that preserves the information (therefore meaning) captured by the HTM Regions.

[3] https://github.com/pulinagrawal/cla-to-mcr/blob/master/cla-mcr.ipynb

This preservation of information happens in both the directions of translation. In the forward translation from sparse distributed representations to Modular Composite Representation vectors we can see that similar sparse distributed representation vectors remained similar when translated to Modular Composite Representation space. This is demonstrated by Figure 5, where an sparse distributed representation vector with less noise, therefore considered similar to the original sparse distributed representation, had a small distance to the translated Modular Composite Representation vector of the original sparse distributed representation after translation. A translated sparse distributed representation vector with more noise, and its corresponding Modular Composite Representation vector in the reference set are far apart in Modular Composite Representation space. In fact, adding 100% noise to a sparse distributed representation, i.e. creating a completely different sparse distributed representation, and translating it to a Modular Composite Representation vector, made it orthogonal to its corresponding reference vector in Modular Composite Representation space. In the backward translation from Modular Composite Representation vectors to sparse distributed representations, we were able to perfectly reconstruct a sparse distributed representation vector that created that Modular Composite Representation vector as evidenced by the perfect precision and recall with zero standard deviation for k=5, as shown in Figure 6. This result depends on the sparsity of the sparse distributed representation vectors, because the sparsity directly affects the number of Modular Composite Representation vectors that are grouped. For a given Modular Composite Representation specification we can only group a certain number of Modular Composite Representation vectors at once. This limit can be increased by increasing the integer range of values that one dimension can hold. For this typical specification of Modular Composite

Representation and sparse distributed representation vectors, we were able to show that all the meaning is preserved in the backward translation as well.

Since the meaning is preserved, we can say that the symbols used by Vector LIDA in the form of Modular Composite Representation vectors are grounded in the environment using the HTM based Sensory Memory. This will allow us to build agents out of Vector LIDA that have grounded symbols. Also, HTM based Sensory Memory allows the agent to represent its environment richly and meaningfully using Sparse Distributed Representations and Modular Composite Representation vectors. Thus, we propose a way to build Sensory Memory within Vector LIDA with HTM Regions and a method of translation, that we show provides faithful representation of the environment, as evidenced by our lossless translation, to compliment the augmented capabilities of Vector LIDA.

CHAPTER 3: Gene Expression Biology: A study using Sparse Autoencoders

By Pulin Agrawal and Bernie J. Daigle Jr.

1. Introduction

We have seen a significant growth in the number of experiments that measure gene expression profiles of various tissues using microarrays. The rapid expansion of the amount of publicly available gene expression data presents opportunities for discovery-driven research into rare diseases with poorly understood etiologies. There have been a lot of studies performed on the data collected from these experiments using various statistical and machine learning techniques. They usually extract a low-dimensional representation of the genomic-data for useful biological discovery (Myers et al., 2006). A lot of these studies consider data from only one experiment, to provide evidence of performance of their classification algorithm. The curse of dimensionality is inherent in such studies because one study that measures gene expression profiles is only able to do the measurement from a few hundred samples at once, while they measure expression of potentially tens of thousands of genes.

A few ways to deal with this problem is to either perform dimensionality reduction techniques on the data to massively reduce the dimensionality of the feature space or to massively increase the amount of data points. Reducing the dimensionality of the feature space often leads to some loss in the signal. Increasing the number of data points is often very difficult because either it entails more data collection, which is very costly, or including data from other experiments. Many studies (Daigle Jr et al., 2010; Tan, Huyck, et al., 2017; Taroni et al., 2018; Tan, Doing, et al., 2017) combine data from few similar experiments to increase the size of their data and

alleviate the curse of dimensionality. Including data from other experiments is not always very straight forward. Different experiments often test different conditions, have different kinds of control groups or use incompatible apparatus for measurements.

The Gene Expression Omnibus (GEO) is an online repository of publicly available gene expression data. It is located at https://www.ncbi.nlm.nih.gov/geo/. It has data from ~100k experiments and a total of 2.8 million samples as of 8th February, 2019. We want to explore the possibility of using this huge amount of publicly available gene expression data to improve the features learned by unsupervised techniques. Even with such a huge number of samples, the ratio of samples to features is not ideal with gene expression data. We have gene expression values from tens of thousands of genes. NIH Library of Integrated Network-Based Cellular Signatures (LINCS) initiative study (Subramanian et al., 2017) has provided a smaller sample of genes that are able to explain most of the variance in gene expression data. We hope that this will help us in further balancing the ratio of samples to features for gene expression data.

The most popular method of dealing with this problem is to pool data from compatible studies and use unsupervised learning methods. A popular method for unsupervised learning is matrix factorization. (Daigle Jr et al., 2010; Taroni et al., 2018)used matrix factorization to identify latent variables in a large compendium of data. They tried to learn features using much bigger datasets comprising of a few tens of thousands of samples. (Taroni et al., 2018) used a transfer learning approach to apply the learned knowledge from this dataset to identify features that aligned well to important biological factors and processes in small datasets of rare diseases. Transfer learning approach is where learned knowledge from one dataset is applied to another

completely different dataset. Some studies that have tried to deal with the issue of curse of dimensionality(Tan, Doing, et al., 2017; Tan, Huyck, et al., 2017)used data from bacterial pathogen Pseudomonas aeruginosa. They applied an unsupervised learning technique using a denoising autoencoder to a dataset with 1,051 samples to show that data-driven gene sets were learned some of which resembled the Kyoto Encyclopedia of Genes and Genomes (KEGG) pathways (Kanehisa & Goto, 2000) and some unidentified gene sets that were differentially active under a given treatment. An autoencoder is trained to reconstruct its input using a feature set learned during training (Bourlard & Kamp, 1988; Liou, Cheng, Liou, & Liou, 2014). In a denoising autoencoder noise is introduced to the training samples during training (Vincent, Larochelle, Bengio, & Manzagol, 2008). This way the network is forced to learn more robust features in the presence of the noise. (Fakoor, Ladhak, Nazi, & Huber, 2013) used sparse autoencoder for classifying samples as being cancerous or not. They conducted multiple classification experiments, each one with a sparse autoencoder network trained on a few hundred to a couple thousand samples. These networks were able to achieve better classification performance on most dataset as compared to other state-of-the-art methods. A sparse autoencoder places an additional constraint of sparsity on the encoding (Ng & others, 2011). We elaborate on this method further since it is relevant to the discussion in this paper.

A sparse autoencoder is a neural network that is trained to reconstruct its input in the output layer using an encoding of the input learned in its hidden layer. It is usually a three-layer network as shown in Figure 9, but there have been proposals to add more layers using a method of training for stacked sparse autoencoders (Xu et al., 2016). The loss function L for a sparse autoencoder has two main components: the reconstruction loss term and the sparsity constraint

term. The reconstruction loss term enforces that the network reconstructs the input as well as possible using L2 loss between the input and output. The sparsity constraint term ensures that the encoding or features used to reconstruct a given input is a sparse set. Thus, the loss L that the neural network tries to minimize during training is given as:

$$L(\boldsymbol{W}, \boldsymbol{h}; \boldsymbol{x}, \boldsymbol{y}) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}\|\boldsymbol{h_i}(\boldsymbol{x_i}).\boldsymbol{W} - \boldsymbol{y_i}\|^2 + \beta.\sum_{j=1}^{s} KL(\rho \parallel \widehat{\boldsymbol{\rho}}_j)$$

where,

W is the variable weight matrix that connects the output layer to the encoding layer. The same weights are used to connect the encoding layer to the input layer.

x is the input matrix with m samples

y is the output matrix of the neural network when the input matrix x is given as input one sample at a time

m is the number of input samples

s is the size of the encoding layer

ρ is the desired sparsity of the encoding layer

h is the encoding at the hidden layer, given by

$$\boldsymbol{h_i}(\boldsymbol{x_i}) = a(\boldsymbol{W}^t.\boldsymbol{x_i})$$

a is a non-linear activation function, typically a sigmoid

$\widehat{\boldsymbol{\rho}}_j$ is the average activity of a unit in the encoding layer, given by

$$\widehat{\boldsymbol{\rho}}_j = \frac{1}{m}\sum_{i=1}^{m}\boldsymbol{W}_j^t.\boldsymbol{x}_i$$

KL is the Kullback-Leibler (KL) divergence between the desired and observed sparsity, given as

$$KL(\rho \parallel \widehat{\boldsymbol{\rho}}_j) = \sum_{j=1}^{s} \rho . log \frac{\rho}{\widehat{\boldsymbol{\rho}}_j} + (1-\rho) . log \frac{1-\rho}{1-\widehat{\boldsymbol{\rho}}_j}$$

β controls the weight of the sparsity constraint term

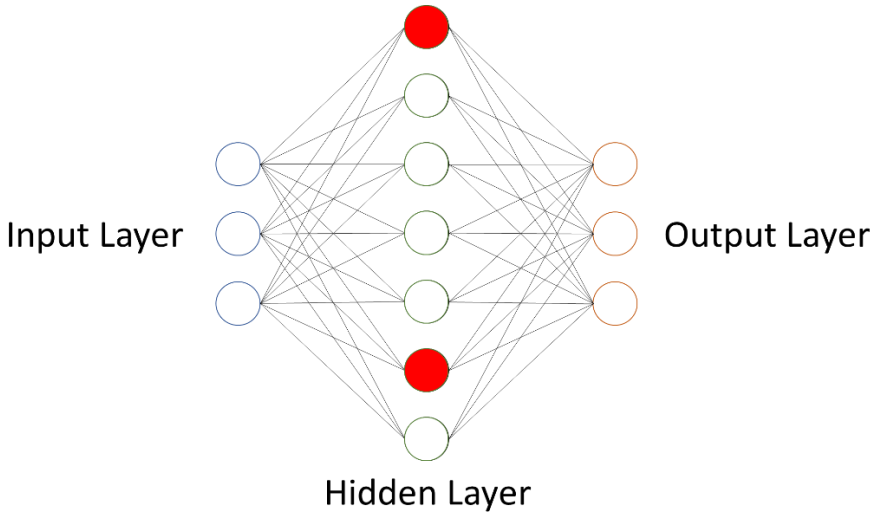


Figure 9 A sparse-autoencoder.

A sparse autoencoder is also able to learn a sparse code of its input using the sparsity constraint on the loss function. Sparse coding is a method for discovering good features automatically using only unlabeled data. Sparse coding is done when an input is represented by a sparse set of features out of many possible features. It has been shown that our brains also perform sparse coding to represent a given stimulus (Ohki et al., 2005). In this paper, we want to use this property of a sparse autoencoder to learn from the huge compendia of samples that we

can obtain from GEO. With the help of feature reduction provided by the LINCS genes, we were able to limit the curse of dimensionality while training this network.

In this paper, we show that the sparse codes learned by a sparse autoencoder trained on a huge set of gene expression samples is able to learn discriminatory representations of groups of samples. We also show that these representations correspond to different statistically significant gene sets between the groups. We describe the pipeline of procedures performed in the Methods section. We provide a detailed account of the results obtained from performing these procedures in the Results sections. Finally, we give a summary and concluding remarks in the Discussion section. Lastly, we end with possible directions that these models can be used in the Future Work section.

2. Methods

In this section, we describe in the detail the steps we performed to conduct the analysis discussed in this paper. Figure 10 gives a brief pictorial representation of the flow of data through the various steps in the process.

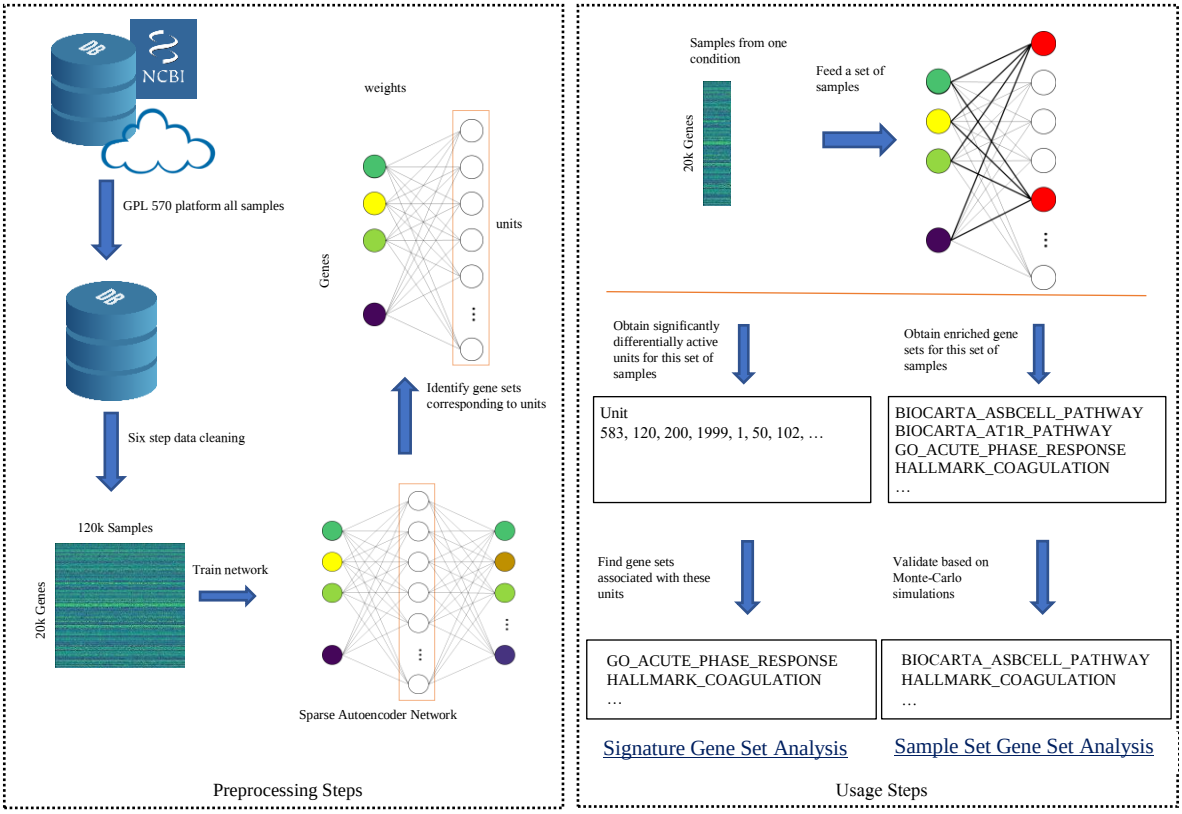


Figure 10 This diagram shows the flow of the data through the various steps from downloading the data from the GEO server to the biological insights produced by our model on various datasets.

2.1. Data Extraction and Cleaning

We downloaded all the gene expression samples from the Gene Expression Omnibus (GEO) (https://www.ncbi.nlm.nih.gov/geo) generated from the GPL570 platform on 7th July, 2017. GPL570 is the GEO accession number for the Affymetrix Human Genome U133 Plus 2.0 Array. We used the getGEO functionfrom R's GEOquery package to retrieve this data. To clean up the data, we performed a battery of procedures to make each sample comparable and suitable to be fed into a neural network. These procedures included the following (in order):

1. The probe expression values returned by the getGEO method were converted to gene expression values by computing the median expression values of all probes matching a gene ID.
2. We removed any sample that had missing values for any gene ID.
3. We check if each sample is log transformed or not. The method used to check for log transform is borrowed from the R code found at https://www.ncbi.nlm.nih.gov/geo/geo2r/. If the sample is not log transformed, we first take the $\log_2$ for each value in the sample.
4. We removed any samples that have NaN values as a result of taking log.
5. We removed any samples that still have a value higher than 1,000,000. We presume that, even after taking log transform, if the value is that high, it means that this is some unwanted artifact in the data.
6. We quantile normalized the samples with respect to each other (code taken from https://github.com/elegant-scipy/notebooks/blob/master/ch2.ipynb)
7. We also scaled the data between 0 to 1 to make it suitable to be fed to a neural network.

2.1.1. Datasets

We extracted the sample ids from two GEO Series (GSE) datasets from the GPL570 platform, GSE15061 (Mills et al., 2009)and GSE8671 (Sabates-Bellver et al., 2007). GSE15061 has three groups of samples from three different conditions Acute Myeloid Leukemia (AML), MyeloDysplastic Syndrome (MDS) and normal. We use only the AML and MDS conditions which have 202 and 164 samples respectively. GSE8671 is a colorectal cancer dataset which has 32 paired samples of cancer and normal tissue from the subjects. Thus, GSE8671 dataset has 32 samples each in the two groups case and control.

2.2. Conventional Limma Gene Set Analysis

This is the most common method of analysis of differential activity between two groups of samples. In this analysis, samples from the two groups are fed into the lmFit function of the 'limma' R package (Ritchie et al., 2015). The results of lmFit are fed in to the eBayes function of 'limma' and finally the resulting p-values of differential activity of genes are used Gene Set Analysis using the runGSA method of the 'piano' R package (Väremo, Nielsen, & Nookaew, 2013) to obtain gene sets relevant to two groups of samples based on FDR scores. A gene set is considered relevant if its FDR score is less than 0.05.

We perform this analysis on the two groups of samples mentioned in the previous section, i.e. GSE15061 and GSE 8671.

2.3. Hyperparameter Tuning

We trained two different Sparse Autoencoder Neural Networks (SANN) implemented in Python using TensorFlow (Abadi et al., 2016). Each of the two networks had one hidden layer and tied weights to the input and output layers. For a loss function, we used the weighted KL-Divergence between the activity of units in hidden layer and the desired sparsity along with the L2 norm of the reconstruction error as defined in Section 1. Sparsity is usually defined as a percentage of hidden units that we expect to be active or have much higher activation than other hidden units for a given input, usually in the range of 1%-10%. This allowed us to train the network for a desired sparsity minimizing the error in reconstruction.

The first SANN contained 1009 input units, each corresponding to one of the LINCS genes present in the cleaned data, i.e. the genes identified by the LINCS program as the ones that capture maximum variance in the Connectivity Map (CMap) dataset (Subramanian et al., 2017). The second SANN contained 21879 input units, corresponding to all available genes in the cleaned data.

We tuned the first SANN with all combinations of the following hyperparameters:

Number of units in hidden layer: 500, 1000 and 2000

Desired sparsity (ρ) of hidden layer: 1%, 2% and 5%

Beta (β): 1 and 3

We tuned the second SANN with all combinations of the following hyperparameters:

Number of units in hidden layer: 1000, 5000, 10000

Desired sparsity (ρ) of hidden layer: 1%, 5%, 10%

Beta (β): 1 and 3

To tune, we randomly split the complete data into a training set containing 80% of the samples and a validation set with the remaining 20%. We created three random splits to assess consistency of the hyperparameter tuning. We then fed each neural network with a random batch of 128 samples from the data until we exhausted the training set. Once such run through of the complete training is considered one epoch. At each epoch we compute the validation reconstruction error (i.e., the error in reconstructing samples in the validation dataset). The

validation reconstruction error reported below represents an average per sample reconstruction error. The training goes on until the validation reconstruction error converges.

We then perform the following analysis in the same way on each of the two SANN models.

2.4. Gene Set Analysis

Our first goal was to determine the extent to which optimal data reconstruction resulted from the discovery of latent biological relationships present in the data. Thus, we performed Gene Set Analysis (GSA) to uncover any such relationships that each network may have learned.
We first collected all the input weights connected to a hidden unit, reversed their signs and normalized them between 0 and 1, and passed them into the runGSA function from R's 'piano' package along with all gene sets from the Molecular Signatures Database v6.1 (MSigDB) (Liberzon et al., 2011). This function allows us to determine whether a particular hidden unit represented any gene sets, as we expect units emulating a gene set to show strong connections (low reversed-sign normalized weights) to many of the genes in a gene set. Specifically, we applied the Wilcoxon rank-sum test in runGSA and treated the normalized absolute values of weights as p-values.

The runGSA function returned a p-value and a False Discovery Rate (FDR) associated with every gene set tested for every hidden unit. We considered all gene sets with FDR ≤ 0.05 as significantly enriched in the corresponding hidden unit.

2.5. Coherence Analysis

We also wanted to measure the degree to which all the gene sets enriched for a particular hidden unit are similar (coherence). For this analysis, we only included biological process gene ontology (GO) (Ashburner et al., 2000) terms from the enriched gene sets to measure the coherence of each hidden unit. We define the coherence score for a hidden unit as the mean pairwise similarity of all enriched GO gene sets for that hidden unit. We calculated similarities, using the "Rel" measure, between two gene sets using the goSim function from the 'GoSemSim' R package (Yu et al., 2010).

Once we obtain these characteristics of the model derived from training, we can perform an analysis of the differences in the way that our model reacts to different groups of samples.

2.6. Signature Gene Set Analysis

We can obtain a signature of the pattern of active hidden units in the model from a group of samples. To obtain this signature, we input each sample to the model and obtain a list of all the units that were most active. Only the x percent most active units of the total number of hidden units were included, where x represents the sparsity of the model. The frequency of the most active units in a given set of samples gives us the signature of that set.

We can use this signature to compare two sets of samples to obtain the differential activity of the most active units. We use the difference of frequency of activation of units between the signature of two sets of samples to obtain the differential activity. To establish significance of the differential activity we perform Monte-Carlo simulations by running the same number of random

samples from our cleaned data as the sets under investigation and obtain a mean(μ) and standard deviation(σ) of differential activity of each unit. These can then be used to obtain an empirical p-value of the observed differential activity of all the units between the given sample sets. The following computation demonstrates how the p-value of the observed differential activity (x_i) was calculated for a unit i.

$$P(\boldsymbol{X_i} \geq x_i|H) = \int_{x_i}^{\infty} f(u)du$$

where,

$\mathbf{X_i}$ is the random variable for the differential activity

H is the null hypothesis that $\mathbf{X_i}$ has the normal distribution $N(\mu_i, \sigma_i^2)$, where μ_i and σ_i are the mean and standard deviation of differential activity of unit i obtained from the Monte-Carlo simulations

f is the probability distribution function of $N(\mu_i, \sigma_i^2)$

We also correct for False Discovery Rate (FDR) since there are thousands of units in a network. The units that have FDR score less than 0.05 were considered differentially enriched between the two groups. We can then look at the gene sets discovered by GSA that correspond to these units.

2.7. Sample Set Gene Set Analysis

Our trained model combined with GSA can be used to obtain enriched gene sets for a set of samples. For each set of samples, we ran each sample through the model and obtained a list of the x% top most active units, where x is the sparsity at which the model was trained. We then

create a list of all the enriched gene sets associated with each of these top most active units. We also keep track of how many times a gene set appeared in the list, this is what we call Gene set Enrichment Count or GEC.

We consider this collection of gene sets as our hypothesis set. We further eliminate any gene sets that do not pass a rigorous statistical test. For this test, we generate 1000 random sets of samples of the same cardinality as a given set of samples. We perform the analysis mentioned above on each of these sets to get an empirical estimate (mean and standard deviation) of GEC of each gene set, as a Monte Carlo simulation. We then compute empirical p-values for the GEC of each gene set in the hypothesis set using a survival function with the empirically estimated mean and standard deviation of that gene set from the Monte Carlo simulation. The following computation demonstrates how the p-value of the observed GEC (gi) was calculated for a gene set i.

$$P(\boldsymbol{G_i} \geq g_i|H) = \int_{g_i}^{\infty} f(u)du$$

where,

$\mathbf{G_i}$ is the random variable for GEC

H is the null hypothesis that $\mathbf{G_i}$ has the distribution $N(\mu_i, \sigma_i^2)$, where μ_i and σ_i are the mean and standard deviation of GEC obtained from the Monte-Carlo simulations

f is the probability distribution function of $N(\mu_i, \sigma_i^2)$

Since we are computing many p-values using this method, we corrected for multiple testing by computing an empirical FDR for each p-value. Empirical FDR of a p-value is given by the

probability of getting the given p-value from the collection of all Monte Carlo p-values that we got. We thus eliminate any gene set from the hypothesis set if it has an FDR greater than 0.05.

The method above is how we can obtain enriched gene sets for a set of samples. These enriched gene sets can then be compared to ones obtained from other sets of samples to identify biological differences between the two sets of samples.

3. Experiments and Results

3.1. Data Extraction and Cleaning

After performing the steps mentioned in the section Data Extraction and Cleaning, we were left with a dataset of ~105k samples with ~21k genes in each. The total size of the data was 20GB. Figure 11 shows a small random subset of these data as a heatmap.

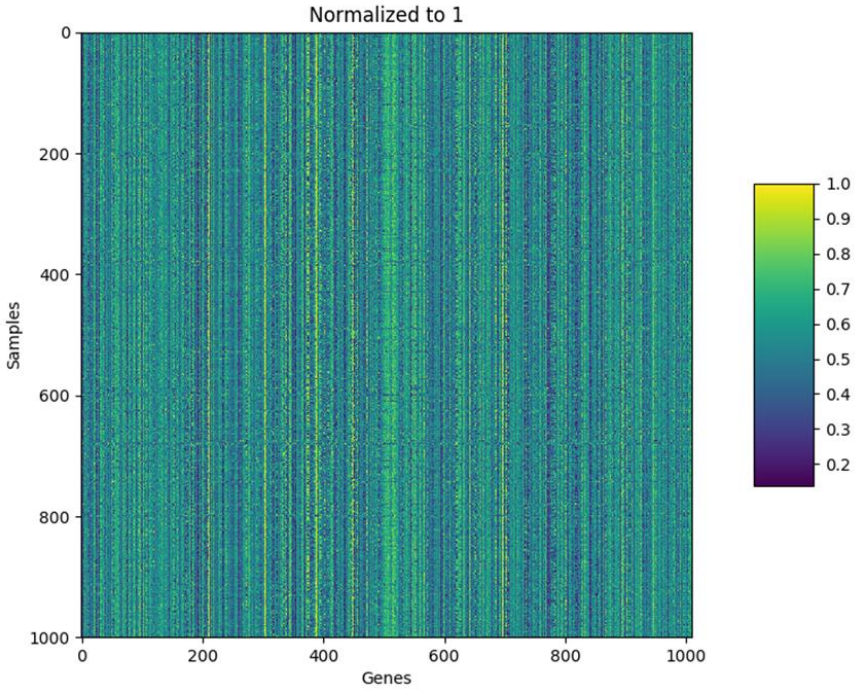


Figure 11 A random subset of processed samples from the GPL570 platform.

3.2. Conventional Limma GSEA Analysis

Performing the conventional Limma GSEA analysis on the data set GSE1506 produced the following list of significantly enriched gene sets with FDR less than 0.05.

ODONNELL_TFRC_TARGETS_DN

AMUNDSON_GAMMA_RADIATION_RESPONSE

BURTON_ADIPOGENESIS_3

CHANG_CYCLING_GENES

CHIANG_LIVER_CANCER_SUBCLASS_PROLIFERATION_UP

CROONQUIST_IL6_DEPRIVATION_DN

CROONQUIST_NRAS_SIGNALING_DN

FISCHER_G2_M_CELL_CYCLE

GNF2_BUB1B

GNF2_CCNA2

GNF2_CCNB2

GNF2_CDC2

GNF2_CDC20

GNF2_CENPE

GNF2_CENPF

GNF2_CKS1B

GNF2_CKS2

GNF2_ESPL1

GNF2_HMMR

GNF2_MKI67

GNF2_PCNA

GNF2_RRM1

GNF2_RRM2

GNF2_SMC4L1

GNF2_TTK

GOLDRATH_ANTIGEN_RESPONSE

GOLDRATH_NAIVE_VS_EFF_CD8_TCELL_DN

GRAHAM_CML_DIVIDING_VS_NORMAL_QUIESCENT_UP

GRAHAM_NORMAL_QUIESCENT_VS_NORMAL_DIVIDING_DN

GSE15750_DAY6_VS_DAY10_EFF_CD8_TCELL_UP

GSE15750_DAY6_VS_DAY10_TRAF6KO_EFF_CD8_TCELL_UP

GSE24634_IL4_VS_CTRL_TREATED_NAIVE_CD4_TCELL_DAY7_UP

GSE24634_TREG_VS_TCONV_POST_DAY7_IL4_CONVERSION_UP

GSE39110_DAY3_VS_DAY6_POST_IMMUNIZATION_CD8_TCELL_DN

HOFFMANN_LARGE_TO_SMALL_PRE_BII_LYMPHOCYTE_UP

KONG_E2F3_TARGETS

LEE_EARLY_T_LYMPHOCYTE_UP

MODULE_198

MODULE_252

MODULE_54

REICHERT_MITOSIS_LIN9_TARGETS

RHODES_UNDIFFERENTIATED_CANCER

ROSTY_CERVICAL_CANCER_PROLIFERATION_CLUSTER

WHITFIELD_CELL_CYCLE_LITERATURE

Performing the conventional Limma GSEA analysis on the data set GSE8671 produced the following list of significantly enriched gene sets with FDR less than 0.05.

BASAKI_YBX1_TARGETS_UP

BLUM_RESPONSE_TO_SALIRASIB_DN

BURTON_ADIPOGENESIS_3

CAIRO_HEPATOBLASTOMA_CLASSES_UP

CHANG_CYCLING_GENES

CHIANG_LIVER_CANCER_SUBCLASS_PROLIFERATION_UP

CROONQUIST_IL6_DEPRIVATION_DN

DODD_NASOPHARYNGEAL_CARCINOMA_DN

DUTERTRE_ESTRADIOL_RESPONSE_24HR_UP

FISCHER_DREAM_TARGETS

GNF2_BUB1B

GNF2_CCNA2

GNF2_CCNB2

GNF2_CDC2

GNF2_CDC20

GNF2_CENPE

GNF2_CENPF

GNF2_CKS1B

GNF2_ESPL1

GNF2_HMMR

GNF2_MCM4

GNF2_MKI67

GNF2_PCNA

GNF2_RRM1

GNF2_RRM2

GOBERT_OLIGODENDROCYTE_DIFFERENTIATION_UP

GOLDRATH_ANTIGEN_RESPONSE

GOLDRATH_EFF_VS_MEMORY_CD8_TCELL_UP

GRADE_COLON_AND_RECTAL_CANCER_UP

GRAHAM_CML_DIVIDING_VS_NORMAL_QUIESCENT_UP

GRAHAM_NORMAL_QUIESCENT_VS_NORMAL_DIVIDING_DN

GSE13547_CTRL_VS_ANTI_IGM_STIM_BCELL_12H_UP

GSE14415_NATURAL_TREG_VS_TCONV_DN

GSE15750_DAY6_VS_DAY10_EFF_CD8_TCELL_UP

GSE15750_DAY6_VS_DAY10_TRAF6KO_EFF_CD8_TCELL_UP

GSE16450_IMMATURE_VS_MATURE_NEURON_CELL_LINE_DN

GSE21063_WT_VS_NFATC1_KO_8H_ANTI_IGM_STIM_BCELL_UP

GSE2405_S_AUREUS_VS_UNTREATED_NEUTROPHIL_DN

GSE24634_TREG_VS_TCONV_POST_DAY7_IL4_CONVERSION_UP

GSE26156_DOUBLE_POSITIVE_VS_CD4_SINGLE_POSITIVE_THYMOCYTE_DN

GSE28726_NAIVE_CD4_TCELL_VS_NAIVE_VA24NEG_NKTCELL_UP

GSE28726_NAIVE_VS_ACTIVATED_CD4_TCELL_DN

GSE39110_DAY3_VS_DAY6_POST_IMMUNIZATION_CD8_TCELL_DN

HALLMARK_E2F_TARGETS

HORIUCHI_WTAP_TARGETS_DN

KINSEY_TARGETS_OF_EWSR1_FLII_FUSION_UP

KOBAYASHI_EGFR_SIGNALING_24HR_DN

KONG_E2F3_TARGETS

LE_EGR2_TARGETS_UP

MARSON_BOUND_BY_E2F4_UNSTIMULATED

MODULE_118

MODULE_403

MODULE_52

MODULE_53

MODULE_54

NAKAYAMA_SOFT_TISSUE_TUMORS_PCA2_UP

ODONNELL_TFRC_TARGETS_DN

POOLA_INVASIVE_BREAST_CANCER_UP

REACTOME_CELL_CYCLE

RHODES_UNDIFFERENTIATED_CANCER

RODRIGUES_THYROID_CARCINOMA_POORLY_DIFFERENTIATED_UP

ROSTY_CERVICAL_CANCER_PROLIFERATION_CLUSTER

RUIZ_TNC_TARGETS_DN

SARRIO_EPITHELIAL_MESENCHYMAL_TRANSITION_UP

SHEDDEN_LUNG_CANCER_POOR_SURVIVAL_A6

SOTIRIOU_BREAST_CANCER_GRADE_1_VS_3_UP

TANG_SENESCENCE_TP53_TARGETS_DN

VECCHI_GASTRIC_CANCER_EARLY_UP

WHITEFORD_PEDIATRIC_CANCER_MARKERS

WHITFIELD_CELL_CYCLE_LITERATURE

WINNEPENNINCKX_MELANOMA_METASTASIS_UP

WONG_EMBRYONIC_STEM_CELL_CORE

ZHAN_MULTIPLE_MYELOMA_PR_UP

3.3. Hyperparameter Tuning

From the hyperparameter tuning performed on the two SANNs using the processed data we found a set of hyperparameters that performed the best based on reconstruction error for each of the two SANNs. The results were as follows.

For the SANN trained on 1009 LINCS genes, we consider the network that had the lowest validation reconstruction loss as the best model and call it the 'lincs_best' model. Figure 12 shows a plot of average reconstruction error of the hyperparameter tuning over the 3 splits of the dataset for this SANN. The 'lincs_best' model at 0.67 reconstruction error has the following hyperparameters:

Number of hidden units-2000

Sparsity-0.05

Beta-1

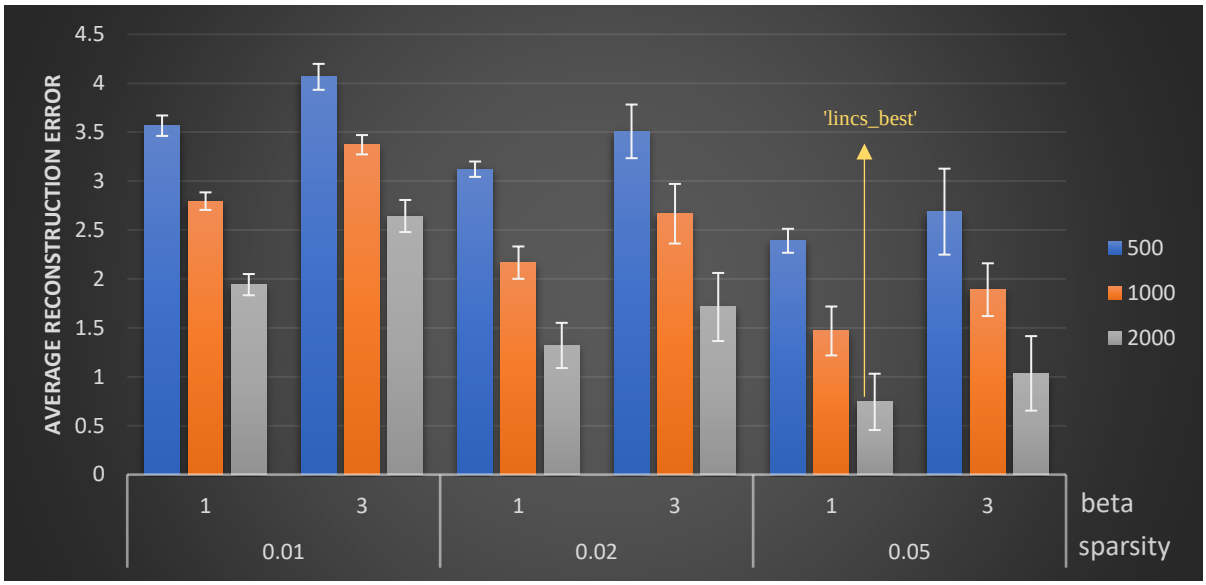


Figure 12 Plot of average reconstruction error for various hyperparameter values for SANN trained on 1009 LINCS genes. Error bars denote standard deviation over 3 random splits of data into training and validation.

For the SANN trained on all the genes, Figure 13 shows a plot of average reconstruction error of the hyperparameter tuning over the 3 splits of the dataset for this SANN.

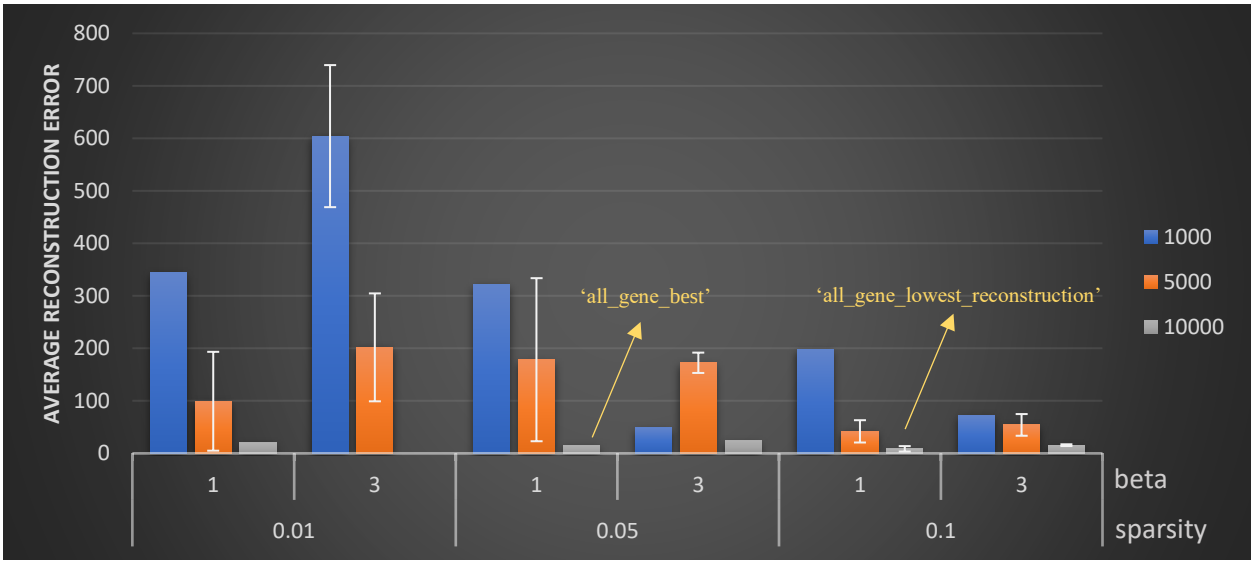


Figure 13 Plot of average reconstruction error for various hyperparameter values. Error bars denote standard deviation over 3 random splits of data into training and validation. Some values are missing because some configurations always resulted in overflow errors during training.

As shown later in the Section 3.5, the model called ‘all_gene_lowest_reconstruction’, with the lowest reconstruction error i.e. 12.317 and number of hidden units=10000, sparsity=0.1 and beta =1, did not perform well on the Coherence Analysis. The model with the second lowest reconstruction error 14.203, with number of hidden units=10000, sparsity=0.05 and beta =1, was then chosen as the ‘all_gene_best’ model based on its performance on Coherence Analysis.

3.4. Gene Set Analysis

We performed Gene Set Analysis on the best models that we obtained from the hyperparameter tuning step, i.e. ‘lincs_best’, ‘all_gene_lowest_reconstruction’, and ‘all_gene_best’. For each of the trained networks we obtained a set of units that had one or more significantly enriched gene sets.

For the ‘lincs_best’ model we had 381 out of 2000 units with one or more gene sets enriched.

For the ‘all_gene_lowest_reconstruction’ model we had 9983 out of 10000 units with one or more gene sets enriched.

For the ‘all_gene_best’ model we had 365 out of 10000 units with one or more gene sets enriched.

3.5. Coherence Analysis

We performed coherence analysis on each of these models. We thus obtain a frequency distribution of coherence of each of the models in terms of number of hidden units. Coherence

frequency distribution for ‘lincs_best’, ‘all_gene_lowest_reconstruction’ and ‘all_gene_best’ model are as shown in Figures 14, 15 and 16 respectively.

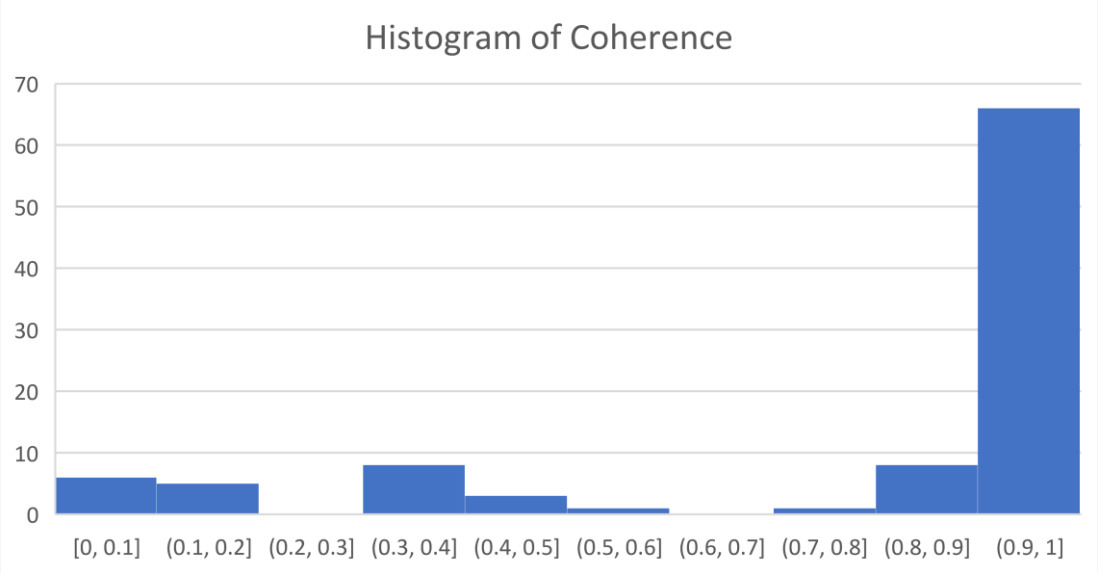


Figure 14 Histogram of coherence of ‘lincs_best’ model's hidden units. X-axis has .1 coherence wide bins and y-axis is the number of units with coherence value in that bin.

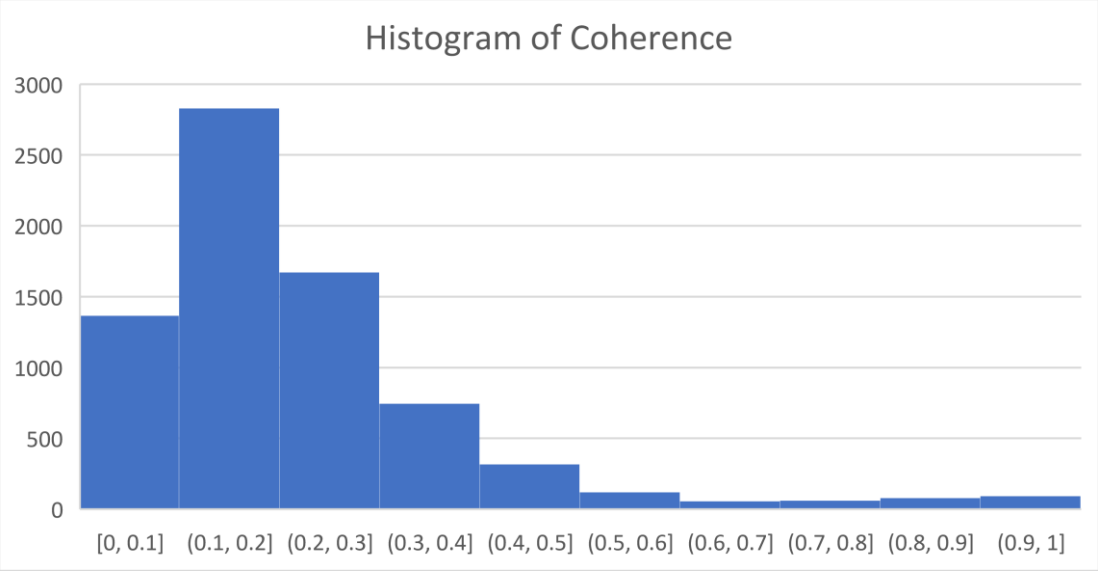


Figure 15 Histogram of coherence of ‘all_gene_lowest_reconstruction’ model's hidden units. X-axis has .1 coherence wide bins and y-axis is the number of units with coherence value in that bin.

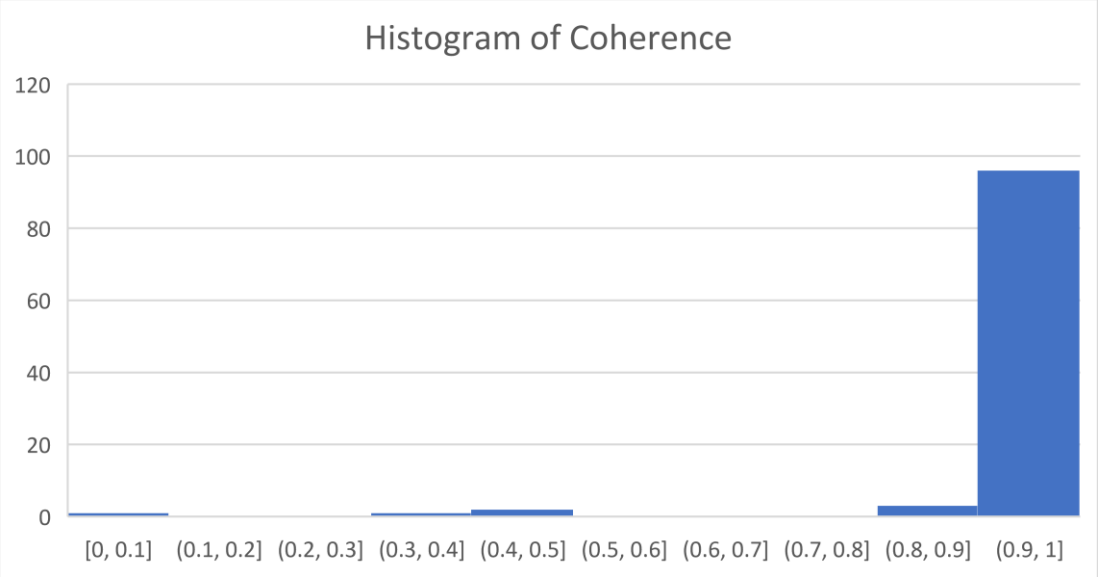


Figure 16 Histogram of coherence of 'all_gene_best' model's hidden units. X-axis has .1 coherence wide bins and y-axis is the number of units with coherence value in that bin.

We also plot, for each of the two models, average coherence of units with a specified cardinality, where cardinality is the number of enriched GO gene sets enriched for that hidden unit. Figure 17, 18 and 19 show average coherence for units with a given cardinality in 'lincs_best', 'all_gene_lowest_reconstruction' and 'all_gene_best' models respectively.

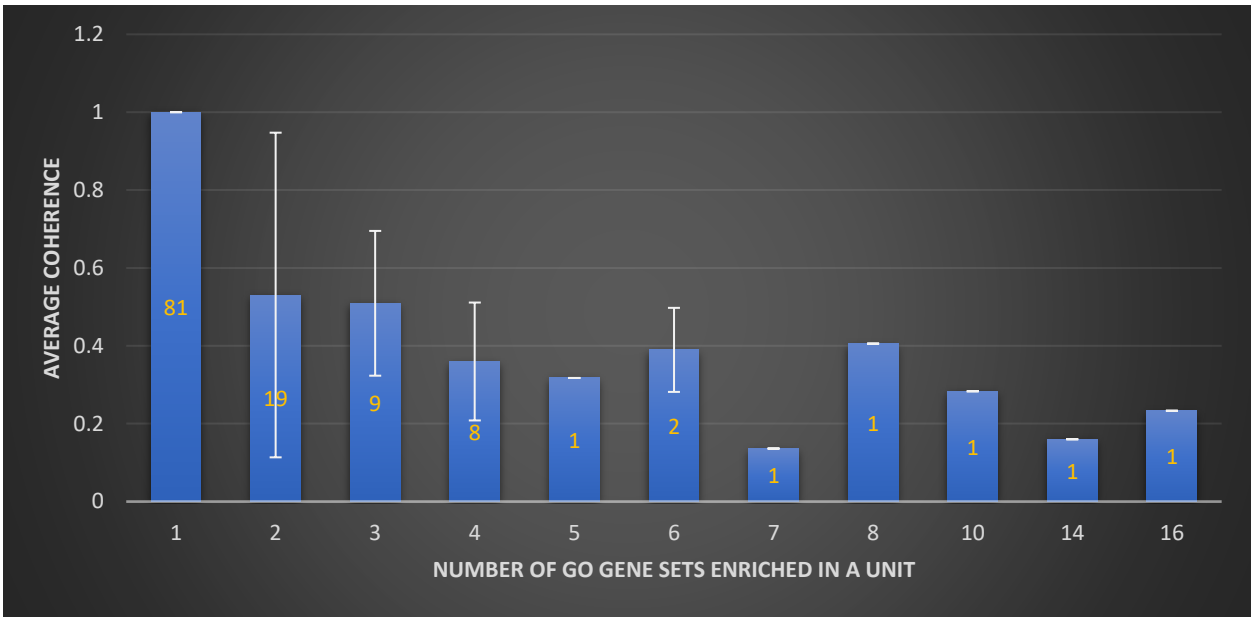


Figure 17 Average coherence of units in 'lincs_best' model with a given cardinality of the number of GO gene sets significantly enriched in them. Values on the bars indicate the number of units with the given cardinality.

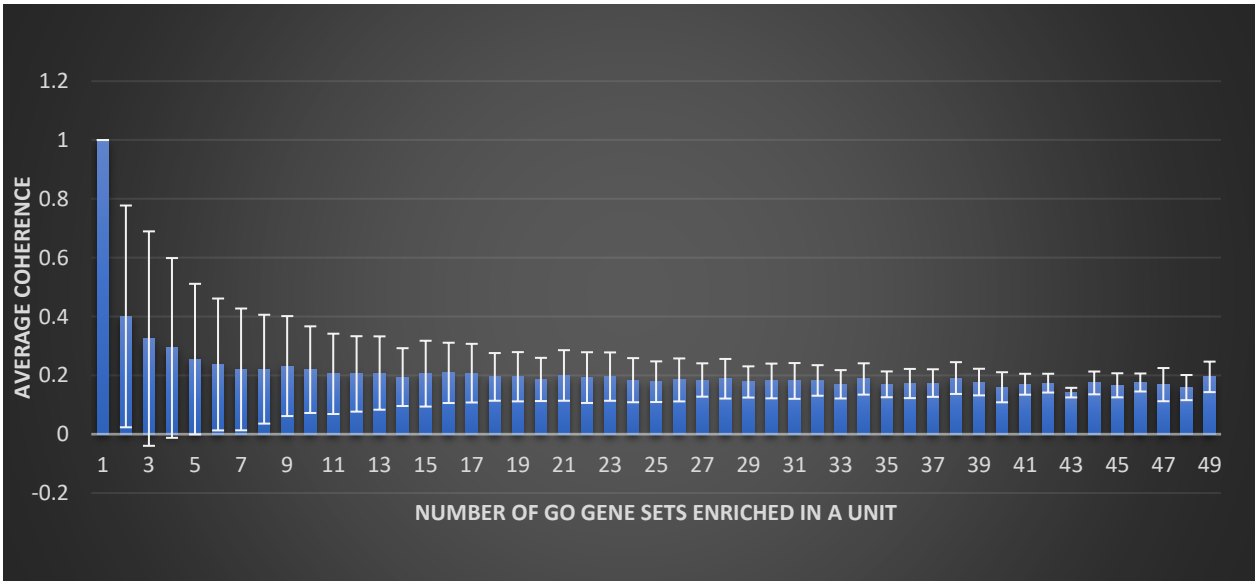


Figure 18 Average coherence of units in 'all_gene_lowest_reconstruction' model with a given cardinality of the number of GO gene sets significantly enriched in them. The number of units with the given cardinality. The number of units with cardinality 1 was 24, 2 was 7 and increasing till 500 for cardinality 7 and then decreasing till 10 at cardinality 59. Cardinalities are not shown for the sake of readability.

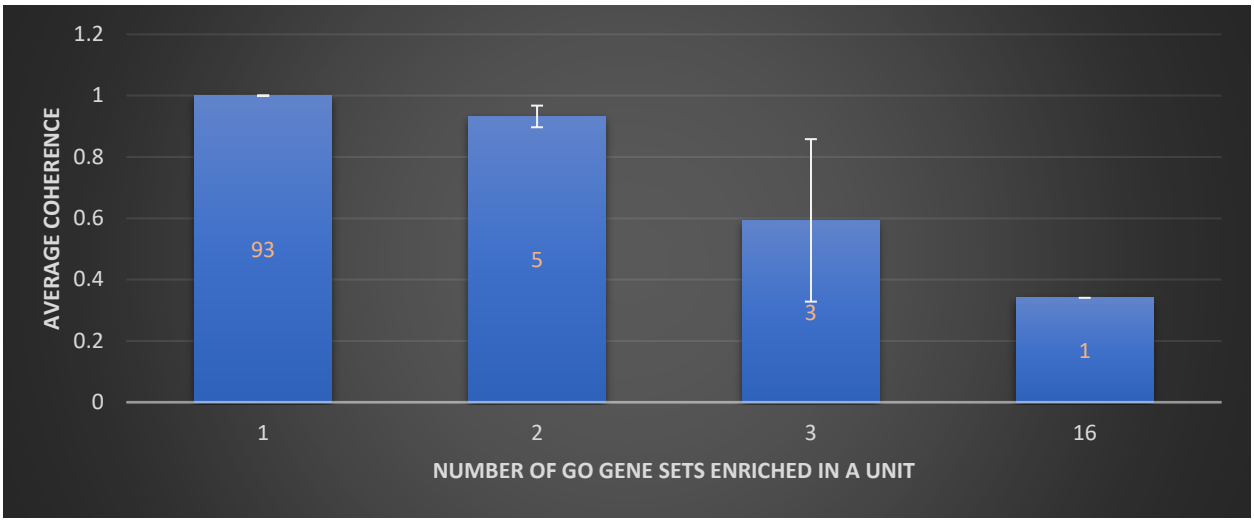


Figure 19 Average coherence of units in 'all_gene_lowest_reconstruction' model with a given cardinality of the number of GO gene sets significantly enriched in them. Values on the bars indicate the number of units with the given cardinality.

We can see in Figure 15 that 'all_gene_lowest_reconstruction' model has poor coherence, with many units having duplicate enriched gene sets. Units with coherence values of 1 were much lower in number in 'all_gene_lowest_reconstruction' as compared to 'all_gene_best' model as we can see in Figures 18 and 19. Therefore, we chose 'all_gene_best' model (along with the 'lincs_best' model) to perform the rest of the analysis.

3.6. Sample Set Signature Analysis

Figure 20 shows the statistics of frequency of activation of units in the two groups of colorectal cancer database (GSE8671) from the 'lincs_best' model.

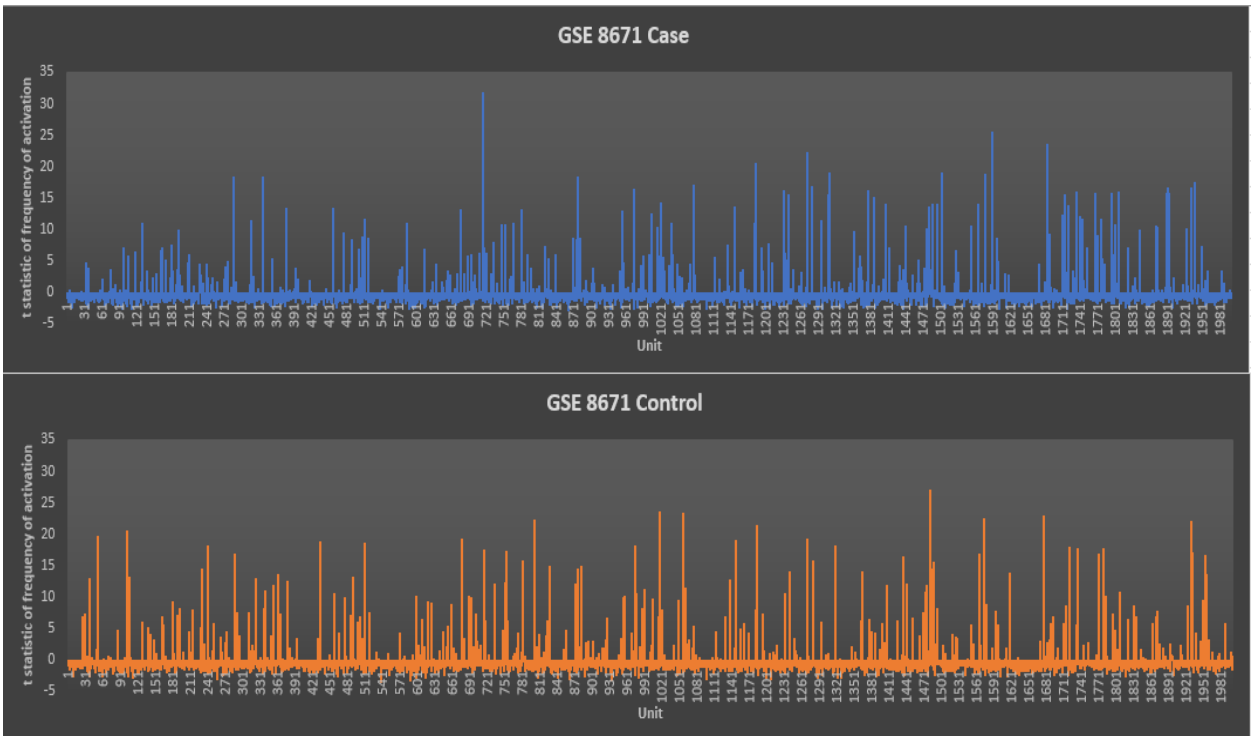


Figure 20 This shows the number of standard deviations away the frequency of activation of a unit is from its mean in a random set of that size in each of the two groups of samples (case and control) of the colorectal cancer dataset (GSE8671).

The gene set analysis performed by measuring differential activity of the frequency of activation as described in Section 2.6, gave us 74 units that were significantly enriched. Table 2 shows the top 5 (based on FDR) of those units out of the significantly differentially active units which had one or more gene sets enriched from the Gene Set Analysis.

Table 2 Top 5 units based on FDR scores from Sample Set Signature Analysis which had one or more gene sets enriched for 'lincs_best' model on GSE8671.

Unit	**Gene Sets**	**FDR**
870	FISCHER_G2_M_CELL_CYCLE	1.53e-57
83	MEISSNER_BRAIN_HCP_WITH_H3K4ME3_AND_H3K27ME3	3.56e-21
427	GSE15750_DAY6_VS_DAY10_TRAF6KO_EFF_CD8_TCELL_UP	5.73e-21
699	GO_CELL_SURFACE, GO_RECEPTOR_ACTIVITY	3.32e-17
463	SOTIRIOU_BREAST_CANCER_GRADE_1_VS_3_UP, VECCHI_GASTRIC_CANCER_EARLY_UP, ODONNELL_TFRC_TARGETS_DN, ODONNELL_TARGETS_OF_MYC_AND_TFRC_DN, TANG_SENESCENCE_TP53_TARGETS_DN, ROSTY_CERVICAL_CANCER_PROLIFERATION_CLUSTER, PUJANA_BRCA2_PCC_NETWORK, MARSON_BOUND_BY_E2F4_UNSTIMULATED, CROONQUIST_IL6_DEPRIVATION_DN, CROONQUIST_NRAS_SIGNALING_DN, NAKAYAMA_SOFT_TISSUE_TUMORS_PCA2_UP, FISCHER_G2_M_CELL_CYCLE, FISCHER_DREAM_TARGETS, GNF2_CDC20, GNF2_CDC2, GNF2_CCNA2, GNF2_CCNB2, GNF2_ESPL1, GNF2_PCNA, GNF2_RFC3, GNF2_RRM1, GNF2_BUB1B, GNF2_HMMR, GNF2_MCM4, GNF2_MKI67, GSE24634_TEFF_VS_TCONV_DAY7_IN_CULTURE_UP, GSE453	1.97e-15

Table 2 (Continued)

Unit	Gene Sets	FDR
1055	MODULE_45, GO_REGULATION_OF_KINASE_ACTIVITY	3.85E-07
1216	GO_NEGATIVE_REGULATION_OF_CELL_PROLIFERATION	8.81E-06
622	GSE29164_UNTREATED_VS_CD8_TCELL_TREATED_ MELANOMA_DAY3_DN	1.96E-05
293	MODULE_27	4.69E-05
548	GO_RESPONSE_TO_BIOTIC_STIMULUS	0.000374
1553	WU_CELL_MIGRATION	0.001881
893	GO_NEURON_DEVELOPMENT, GO_NEURON_PROJECTION_DEVELOPMENT, GO_CELL_PROJECTION_ORGANIZATION, GO_CELL_DEVELOPMENT	0.004628
966	GO_CELLULAR_RESPONSE_TO_LIPID, GO_RESPONSE_TO_LIPID	0.005335
924	MEISSNER_BRAIN_HCP_WITH_H3K4ME3_AND_H3K27ME3, GO_EXTRACELLULAR_SPACE	0.008318
240	SMID_BREAST_CANCER_BASAL_UP	0.012108
232	MODULE_342	0.036145

Twenty one gene sets out of a total of 53 present in the significantly differentially active units overlapped with the enriched gene sets from the Conventional Limma Gene Set Analysis for this dataset.

Figure 21 shows the statistics of frequency of activation of units in the two groups (AML and MDS) of the leukemia dataset (GSE15061) using the 'lincs_best' model.

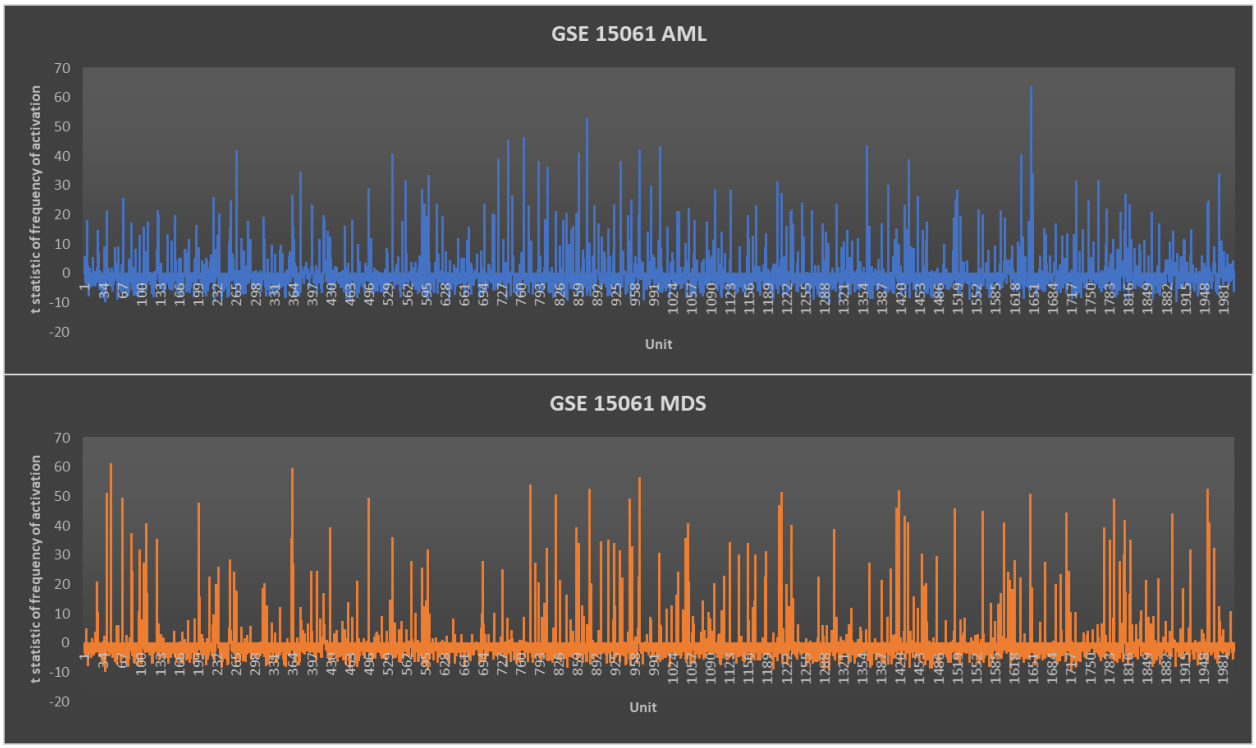


Figure 21 This shows the number of standard deviations away the frequency of activation of a unit is from its mean in a random set of that size in each of the two groups of samples (MyeloDysplastic Syndrome (MDS) and acute myeloid leukemia (AML)) from the leukemia dataset (GSE 15061).

The gene set analysis performed by measuring differential activity of the frequency of activation as described in Section 2.6, gave us 283 units that were significantly differentially active. Table 3 shows top 5 units out of the significantly differently active units which had one or more gene sets enriched from the Gene Set Analysis. Rest of the units are in Table 10 in the Supplementary Material section. Two gene sets out of a total of 50 present in the significantly differentially active units overlapped with the enriched gene sets from the Conventional Limma Gene Set Analysis for this dataset.

Table 3 Top 5 units ordered by FDR scores from Sample Set Signature Analysis which had one or more gene sets enriched for 'lincs_best' model on GSE15061.

Unit	**Gene Sets**	**FDR**
1208	CHARAFE_BREAST_CANCER_LUMINAL_VS_BASAL_DN	< 1e-300
1300	KEGG_MAPK_SIGNALING_PATHWAY, GO_SIGNAL_TRANSDUCER_ACTIVITY	< 1e-300
1446	GO_REGULATION_OF_CELLULAR_COMPONENT_MOVEMENT	< 1e-300
953	GAL_LEUKEMIC_STEM_CELL_DN, LEE_EARLY_T_LYMPHOCYTE_UP, GSE36476_CTRL_VS_TSST_ACT_72H_MEMORY_CD4_TCELL_YOUNG_DN	< 1e-300
638	KIM_WT1_TARGETS_UP	8.3e-286

We did not show statistics of frequency of activation plots for the 'all_gene_best' model, due to it having too many units.

Table 4 shows all the gene sets from the Signature Gene Set Analysis when the two groups of samples from GSE8671 colorectal cancer dataset were fed into 'all_gene_best' model.

Table 4 Significantly differentially active units ordered by FDR scores from Sample Set Signature Analysis which had one or more gene sets enriched for 'all_gene_best' model on GSE8671.

Unit	**Gene Sets**	**FDR**
7731	GO_MYELOID_CELL_HOMEOSTASIS	2.37e-38
5142	GSE24634_TEFF_VS_TCONV_DAY5_IN_CULTURE_UP	3.46e-28
94	GO_CELLULAR_IRON_ION_HOMEOSTASIS	5.01e-26
7672	RNCTGNYNRNCTGNY_UNKNOWN	7.51e-24
6690	GO_REGULATION_OF_GENERATION_OF_PRECURSOR_METABOLITES_AND_ENERGY	7.74e-13
8801	MOHANKUMAR_TLX1_TARGETS_DN, GSE45881_CXCR6HI_VS_CXCR1LO_COLONIC_LAMINA_PROPRIA_DN	1.25e-10
2047	chr17q12	8.77e-10
7048	GSE33425_CD161_HIGH_VS_INT_CD8_TCELL_UP	1.37e-07
7173	WANG_TARGETS_OF_MLL_CBP_FUSION_DN	1.96e-07
7149	SNF5_DN.V1_DN	1.16e-06
3744	HUANG_GATA2_TARGETS_DN	2.55e-05
5321	REACTOME_METABOLISM_OF_RNA	0.0001
2914	GO_TRANSLATIONAL_INITIATION	0.0001
2638	GO_MOTOR_NEURON_AXON_GUIDANCE	0.0102
6201	SCHLOSSER_MYC_TARGETS_AND_SERUM_RESPONSE_UP	0.0137

Table 5 shows all the gene sets from the Signature Gene Set Analysis when the two groups of samples from GSE15061 leukemia dataset were fed into 'all_gene_best' model.

Table 5 Significantly differentially active units ordered by FDR scores from Sample Set Signature Analysis which had one or more gene sets enriched for 'all_gene_best' model on GSE15061.

Unit	**Gene Sets**	**FDR**
755	GSE18281_CORTEX_VS_MEDULLA_THYMUS_UP	3.54e-174
2638	GO_MOTOR_NEURON_AXON_GUIDANCE	1.05e-106
94	GO_CELLULAR_IRON_ION_HOMEOSTASIS	1.82e-83
1158	GSE25088_WT_VS_STAT6_KO_MACROPHAGE_IL4_STIM_UP	4.95e-70
667	GO_ACTIN_FILAMENT_BINDING	2.64e-60
5342	GSE2770_TGFB_AND_IL4_VS_IL12_TREATED_ACT_CD4_TCELL_48H_DN	2.19e-53
7610	BUYTAERT_PHOTODYNAMIC_THERAPY_STRESS_DN	1.57e-51
6891	GO_GLYCOLIPID_BINDING	2.71e-38
2914	GO_TRANSLATIONAL_INITIATION	5.47e-25
9614	GO_RESPONSE_TO_BACTERIUM, GO_NEGATIVE_REGULATION_OF_INFLAMMATORY_RESPONSE, GO_NEGATIVE_REGULATION_OF_RESPONSE_TO_WOUNDING	2.58e-12
5647	GO_CELLULAR_RESPONSE_TO_EXTRACELLULAR_STIMULUS	5.46e-05
3250	RAY_TUMORIGENESIS_BY_ERBB2_CDC25A_UP	6.91e-05
9334	BOYLAN_MULTIPLE_MYELOMA_D_UP	9.58e-05

3.7. Sample Set Gene Set Analysis

We then ran the sets of samples from the two conditions, i.e. case and control tested in GSE 8671 using the Sample Set Gene Set Analysis on the 'lincs_best' model. The following Tables 6 and 7 contain the gene sets unique to each condition and a set of gene sets common to the two conditions and the list of common gene sets follow it.

Table 6 Unique gene sets from Sample Set Gene Set Analysis on case samples of GSE 8671 from 'lincs_best' model.

Gene set	**FDR**
KEGG_BLADDER_CANCER	0.004474273
GO_REGULATION_OF_BODY_FLUID_LEVELS	0.011185682
WU_CELL_MIGRATION	0.029082774
GO_RESPONSE_TO_TOXIC_SUBSTANCE	0.035794183
GO_SIGNALING_RECEPTOR_ACTIVITY	0.03803132
GO_NEGATIVE_REGULATION_OF_MULTICELLULAR_ORGANISMAL_PROCESS	0.046979866
MORI_IMMATURE_B_LYMPHOCYTE_DN	0.049217002

Table 7 Unique gene sets from Sample Set Gene Set Analysis on control samples of GSE 8671 from 'lincs_best' model.

Gene set	**FDR**
CAIRO_HEPATOBLASTOMA_DN	0.024608501
MODULE_212	0.024608501
GO_SMALL_MOLECULE_CATABOLIC_PROCESS	0.024608501
HOSHIDA_LIVER_CANCER_SUBCLASS_S3	0.024608501
GO_SMALL_MOLECULE_BIOSYNTHETIC_PROCESS	0.024608501
GO_SMALL_MOLECULE_METABOLIC_PROCESS	0.024608501
HSIAO_LIVER_SPECIFIC_GENES	0.024608501
GO_RESPONSE_TO_OXYGEN_CONTAINING_COMPOUND	0.031319911
GO_RESPONSE_TO_ENDOGENOUS_STIMULUS	0.035794183
GO_RESPONSE_TO_HORMONE	0.040268456
GO_RESPONSE_TO_ORGANIC_CYCLIC_COMPOUND	0.040268456
GO_SIGNAL_TRANSDUCER_ACTIVITY	0.046979866
GO_RESPONSE_TO_LIPID	0.049217002

Common

TATAAA_TATA_01

STAT5A_01

GGGTGGRR_PAX4_03

GO_INFLAMMATORY_RESPONSE

KEGG_PATHWAYS_IN_CANCER

GO_DEFENSE_RESPONSE

We then ran the sets of samples from the two conditions, i.e. AML tissue and MDS tissue, tested in GSE15061 using the Sample Set Gene Set Analysis on the 'lincs_best' model. Tables 8 an 9 show the gene sets unique to each condition, followed by a list of gene sets common to the two conditions.

Table 8 Unique gene sets from Sample Set Gene Set Analysis on MDS samples of GSE 15061 from 'lincs_best' model.

Gene set	**FDR**
GNF2_RAD23A	0.020134
GSE7852_LN_VS_THYMUS_TCONV_DN	0.020134
GSE27241_WT_VS_RORGT_KO_TH17_POLARIZED_CD4_TCELL_UP	0.020134
GNF2_BNIP3L	0.020134
BENPORATH_SUZ12_TARGETS	0.020134
GNF2_MAP2K3	0.020134
GO_REGULATION_OF_BODY_FLUID_LEVELS	0.022371
STK33_DN	0.024609
GSE22886_NAIVE_TCELL_VS_MONOCYTE_UP	0.029083
GO_RESPONSE_TO_LIPID	0.035794
GSE13547_2H_VS_12_H_ANTI_IGM_STIM_BCELL_UP	0.042506

Table 9 Unique gene sets from Sample Set Gene Set Analysis on AML samples of GSE 15061 from 'lincs_best' model.

Gene set	**FDR**
HALLMARK_COAGULATION	0.022371365
GO_CELLULAR_RESPONSE_TO_EXTRACELLULAR_STIMULUS	0.022371365
NAGASHIMA_NRG1_SIGNALING_UP	0.022371365
GSE36891_POLYIC_TLR3_VS_PAM_TLR2_STIM_PERITONEAL_MACROPHAGE_UP	0.022371365
FRASOR_RESPONSE_TO_SERM_OR_FULVESTRANT_DN	0.022371365
SERVITJA_LIVER_HNF1A_TARGETS_UP	0.022371365
ESC_J1_UP_EARLY.V1_DN	0.022371365
MODULE_44	0.035794183
MODULE_1	0.042505593
MODULE_2	0.044742729
TARTE_PLASMA_CELL_VS_PLASMABLAST_UP	0.049217002

Common

ERB2_UP.V1_UP

KRAS.600_UP.V1_UP

GO_CELLULAR_RESPONSE_TO_LIPID

GO_PLASMA_MEMBRANE_PROTEIN_COMPLEX

MODULE_6

MODULE_88

GO_CELL_PROJECTION

MODULE_38

MODULE_220

LIM_MAMMARY_STEM_CELL_UP

This analysis did not give any significantly enriched gene sets for ‘all_gene_best’ model for either of the two datasets.

4. Conclusion and Discussion

The hyperparameters generally performed better in terms of reconstruction error if we increased the number of hidden layers. It is computationally expensive to train such networks. Increasing the number of hidden layers will certainly exacerbate this problem. Obtaining results from larger network will take more time. Although, we may need to obtain results from larger networks, we believe the results from these smaller networks are also worth reporting.

The conventional limma Gene Set analysis from the GSE 8671 dataset gave us 73 significantly enriched gene sets, 10 of which were related to cancer in various parts of the body. GRADE_COLON_AND_RECTAL_CANCER_UP is one example of a gene set which is directly relevant to colorectal cancer.

The conventional limma Gene Set analysis from the GSE 15061 dataset gave us 44 significantly enriched gene sets, 3 of which were the following cancer related sets: CHIANG_LIVER_CANCER_SUBCLASS_PROLIFERATION_UP, RHODES_UNDIFFERENTIATED_CANCER and ROSTY_CERVICAL_CANCER_PROLIFERATION_CLUSTER.

The hyperparameter tuning results show us that networks with more hidden units and higher sparsity generally performed better. The hyperparameter beta showed a peculiar behavior, where a higher value of beta was worse for the 1009 input network and a lower value was better for the 20k gene network. The histogram plots in Figures 4 and 5 from the coherence analysis showed that the distribution of coherence was bi-modal, peaking at around .2 and at 1.

Sample set signature analysis showed that the two groups had different patterns of activity in the network, indicating that the network is able to capture the differences between the groups. This is especially encouraging because this suggests that the activity of the hidden units in this network can be used for purposes of classification of samples between the two groups.

The gene sets from the Signature Gene Set Analysis on the GSE 8671 colorectal cancer dataset using the 'lincs_best' model gave several cancer-related gene sets, including unit 463 with terms like SOTIRIOU_BREAST_CANCER_GRADE_1_VS_3_UP, VECCHI_GASTRIC_CANCER_EARLY_UP, ROSTY_CERVICAL_CANCER_PROLIFERATION_CLUSTER and PUJANA_BRCA2_PCC_NETWORK.

The gene sets from the Signature Gene Set Analysis on the GSE 15061 leukemia dataset using the 'lincs_best' model also gave a few cancer-related gene sets such as CHARAFE_BREAST_CANCER_LUMINAL_VS_BASAL_DN, KEGG_MAPK_SIGNALING_PATHWAY and GAL_LEUKEMIC_STEM_CELL_DN. GAL_LEUKEMIC_STEM_CELL_DN is a gene set directly related to leukemia that appears with the leukemia data set .

The gene sets from the Signature Gene Set Analysis on the GSE 8671 colorectal cancer dataset using the 'all_gene_best' model produced a few cancer-related gene sets such as MOHANKUMAR_TLX1_TARGETS_DN and HUANG_GATA2_TARGETS_DN, along with a gene set GSE45881_CXCR6HI_VS_CXCR1LO_COLONIC_LAMINA_PROPRIA_DN that is involved in immune response in colon.

We can observe from the Sample Set Gene Set Analysis from the colorectal cancer dataset (GSE 8671) that the KEGG_PATHWAYS_IN_CANCER gene set was active in both the samples. This is interesting because in the GSE8671 dataset the two groups of samples are taken from the same subject and thus we might expect to see some cancer pathways active in both the groups. Also, the two GO gene sets GO_INFLAMMATORY_RESPONSE and GO_DEFENSE_RESPONSE are not surprising to see, given that we might expect subjects in an ailing condition to undergo inflammatory and defense response processes. We also see gene sets like KEGG_BLADDER_CANCER, GO_REGULATION_OF_BODY_FLUID_LEVELS, WU_CELL_MIGRATION and GO_RESPONSE_TO_TOXIC_SUBSTANCE which may also be indicative of cancer tissue.

In the Sample Set Gene Set Analysis from the leukemia dataset (GSE 15061), we saw many gene sets related to blood showing up in both conditions. We saw some gene sets in the AML condition like HALLMARK_COAGULATION and TARTE_PLASMA_CELL_VS_PLASMABLAST_UP that can be implicated in blood-related disorders.

5. Future Work

We conclude that sparse autoencoder is capable of discovering important biological features in the data. These models are important to keep around because of the nature of unsupervised learning approach that we used. Since no labels were used to train these models, they can be updated whenever we encounter new experiments by training these networks further on new data from these experiments.

An obvious an interesting next step would be to see classification results from using the activations of hidden units from two groups of samples to train a classifier. Another challenging but a fruitful endeavor would be to look at various units that did not correspond to a pre-discovered gene set and map the genes that they are heavily connected to. This can then be used to identify many undiscovered gene sets.

Since we can see different signatures of activity from two groups of samples, we can trace back the activity of the units to the genes that most influence the activity of a unit and thus find undiscovered gene sets that respond differentially to the two groups, which can help in identifying crucial features of a condition and potential treatments of diseases.

6. Supplementary Material

Table 10 Remaining units from 'lincs_best' model's Sample Set Signature Analysis on GSE15061

Unit	Gene Sets	FDR
1446	GO_REGULATION_OF_CELLULAR_COMPONENT_MOVEMENT	1.57E-264
107	GO_RECEPTOR_ACTIVITY	4.14E-207
1846	PLASARI_TGFB1_TARGETS_10HR_UP	7.50E-177
1577	KEGG_PATHWAYS_IN_CANCER, KEGG_BLADDER_CANCER	1.89E-54
1072	SMID_BREAST_CANCER_BASAL_UP	3.01E-43
239	MODULE_47	1.41E-42
199	MODULE_342	1.81E-36
1722	BUYTAERT_PHOTODYNAMIC_THERAPY_STRESS_UP	9.11E-35
671	WONG_ADULT_TISSUE_STEM_MODULE	7.96E-34
218	GO_POSITIVE_REGULATION_OF_BIOSYNTHETIC_PROCESS, GO_POSITIVE_REGULATION_OF_GENE_EXPRESSION, GO_MUSCLE_STRUCTURE_DEVELOPMENT	4.81E-33
1416	SCHUETZ_BREAST_CANCER_DUCTAL_INVASIVE_UP, GO_BLOOD_VESSEL _MORPHOGENESIS, GO_ANGIOGENESIS	8.55E-31
856	ONDER_CDH1_TARGETS_2_DN	4.99E-28
699	GO_CELL_SURFACE, GO_RECEPTOR_ACTIVITY	3.38E-20
1016	GSE5589_UNSTIM_VS_45MIN_LPS_AND_IL6_STIM_MACROPHAGE_UP	2.02E-18
108	ESC_V6.5_UP_EARLY.V1_DN	6.32E-18

Table 10 (Continued)

Unit	Gene Sets	FDR
427	GSE15750_DAY6_VS_DAY10_TRAF6KO_EFF_CD8ELL_UP	9.88E-17
1151	KINSEY_TARGETS_OF_EWSR1_FLII_FUSION_UP	6.22E-16
1926	GOZGIT_ESR1_TARGETS_DN	1.05E-15
1053	MANALO_HYPOXIA_UP	1.25E-13
332	GO_FORMATION_OF_PRIMARY_GERM_LAYER	9.91E-13
90	KRIGE_RESPONSE_TO_TOSEDOSTAT_6HR_UP	1.35E-10
1809	GO_PROTEINACEOUS_EXTRACELLULAR_MATRIX	1.70E-10
588	WONG_ADULT_TISSUE_STEM_MODULE, KOINUMA_TARGETS_OF_SMAD2 _OR_SMAD3	1.72E-10
92	GAL_LEUKEMIC_STEM_CELL_DN, ODONNELL_TFRC_TARGETS_DN, GRAHAM_CML_DIVIDING_VS_NORMAL_QUIESCENT_UP, GRAHAM_NORMAL_QUIESCENT_VS_NORMAL_DIVIDING_DN , ROSTY_CERVICAL_CANCER_PROLIFERATION_CLUSTER, GAVIN_FOXP3_TARGETS_CLUSTER_P6, HELLER_SILENCED_BY_METHYLATION_UP, WHITEFORD_PEDIATRIC_CANCER_MARKERS, CROONQUIST_IL6_DEPRIVATION_DN, POOLA_INVASIVE_BREAST_CANCER_UP, CHIANG_LIVER_CANCER_SUBCLASS_PROLIFERATION_UP,	4.19E-08

Table 10 (Continued)

Unit	Gene Sets	FDR
	NAKAYAMA_SOFT_TISSUE_TUMORS_PCA2_UP, WHITFIELD_CELL_CYCLE_LITERATURE, GNF2_CDC20, GNF2_CDC2, GNF2_CCNA2, GNF2_CCNB2, GNF2_ESPL1, GNF2_RRM2, GNF2_CKS2, GNF2_HMMR, GNF2_MKI67, GNF2_TTK, MODULE_54, MODULE_118, GO_REGULATION_OF_NUCLEAR_DIVISION, GSE15750_DAY6_VS_DAY10_EFF_CD8_TCELL_UP, GSE2405_S_AUREUS_VS_UNTREATED_NEUTROPHIL_DN, GSE25088_WT_VS_STAT6_KO_MACROPHAGE_IL4_STIM_DN, GSE32164_RESTING_DIFFERENTIATED_VS_ALTERNATIVELY_ACT_M2_MACROPHAGE_UP, GSE33424_CD161_HIGH_VS_INT_CD8_TCELL_DN, GSE39110_DAY3_VS_DAY6_POST_IMMUNIZATION_CD8_TCELL_DN, GSE45365_WT_VS_IFNAR_KO_CD11B_DC_MCMV_INFECTION_DN, GSE45365_HEALTHY_VS_MCMV_INFECTION_CD11B_DC_DN	
775	GSE10325_CD4_TCELL_VS_BCELL_UP	1.65E-07
50	ONDER_CDH1_TARGETS_2_DN, HSIAO_LIVER_SPECIFIC_GENES,	7.10E-07

Table 10 (Continued)

Unit	Gene Sets	FDR
	HOSHIDA_LIVER_CANCER_SUBCLASS_S3, CAIRO_HEPATOBLASTOMA _DN, MODULE_212, GO_SMALL_MOLECULE_METABOLIC_PROCESS, GO_SMALL_MOLECULE_BIOSYNTHETIC_PROCESS, GO_SMALL_MOLECULE_CATABOLIC_PROCESS, RAF_UP.V1_DN	
1553	WU_CELL_MIGRATION	8.57E-07
487	GO_INFLAMMATORY_RESPONSE	9.35E-07
1709	GO_REGULATION_OF_BODY_FLUID_LEVELS	1.26E-06
74	GO_RECEPTOR_ACTIVITY	3.53E-06
1184	HALLMARK_ESTROGEN_RESPONSE_EARLY	6.33E-06
1161	GO_REGULATION_OF_EPITHELIAL_CELL_PROLIFERATION, GO_POSITIVE_REGULATION_OF_MULTICELLULAR_ORGANISMAL_PROCESS	1.57E-05
1166	MODULE_75	1.63E-05
624	ODONNELL_TFRC_TARGETS_DN, ROSTY_CERVICAL_CANCER_PROLIFERATION_CLUSTER, GNF2_CDC20, GNF2_CDC2, GNF2_CCNA2, GNF2_HMMR, GSE15750_DAY6_VS_DAY10_TRAF6KO_EFF_CD8_TCELL_UP	5.87E-05

Table 10 (Continued)

Unit	Gene Sets	FDR
697	GSE11961_MEMORY_BCELL_DAY7_VS_PLASMA_CELL_DAY7_UP, GSE43863_LY6C_INT_CXCR5POS_VS_LY6C_LOW_CXCR5NEG_EFFECTOR_CD4_TCELL_UP	0.0001
1298	SWEET_KRAS_TARGETS_UP	0.000134
816	BENPORATH_ES_WITH_H3K27ME3, PHONG_TNF_RESPONSE_VIA_P38_PARTIAL, GO_REGULATION_OF_MULTICELLULAR_ORGANISMAL_DEVELOPMENT, GO_REGULATION_OF_CELL_DIFFERENTIATION, GO_RESPONSE_TO_BIOTIC_STIMULUS	0.00053
966	GO_CELLULAR_RESPONSE_TO_LIPID, GO_RESPONSE_TO_LIPID	0.000612
1514	WONG_ADULT_TISSUE_STEM_MODULE	0.000989
336	GO_SIGNALING_RECEPTOR_ACTIVITY	0.001593
301	NABA_MATRISOME, MODULE_220, GO_PROTEINACEOUS_EXTRACELLULAR_MATRIX	0.002024
1828	VART_KSHV_INFECTION_ANGIOGENIC_MARKERS_UP, VART_KSHV_INFECTION_ANGIOGENIC_MARKERS_DN	0.002025
94	GSE22886_DC_VS_MONOCYTE_DN	0.002507
449	RODWELL_AGING_KIDNEY_UP, MODULE_2	0.011659
494	GSE35825_UNTREATED_VS_IFNG_STIM_MACROPHAGE_UP	0.015255

Table 10 (Continued)

Unit	Gene Sets	FDR
1924	GO_NEGATIVE_REGULATION_OF_MULTICELLULAR_ORGANISMAL_PROCESS	0.017982
235	DUTERTRE_ESTRADIOL_RESPONSE_24HR_DN	0.019782
420	CHICAS_RB1_TARGETS_CONFLUENT, GSE9988_ANTI_TREM1_VS_CTRL_TREATED_MONOCYTES_UP, GSE9988_ANTI_TREM1_VS_VEHICLE_TREATED_MONOCYTES_UP	0.022893
1081	CHARAFE_BREAST_CANCER_LUMINAL_VS_BASAL_DN, WONG_ADULT_TISSUE_STEM_MODULE, GO_REGULATION_OF_COAGULATION, GO_NEGATIVE_REGULATION_OF_CELL_DEATH, GO_GROWTH, GO_REGULATION_OF_CELL_DEATH, GO_NEGATIVE_REGULATION_OF_RESPONSE_TO_WOUNDING, GO_TISSUE_DEVELOPMENT, GO_NEGATIVE_REGULATION_OF_RESPONSE_TO_EXTERNAL_STIMULUS, GSE24634_TEFF_VS_TCONV_DAY7_IN_CULTURE_DN	0.047687

CHAPTER 4: Conclusion

1. Summary of Conclusions

Using right representations are important to solve a problem. Sparse representation is useful way of representing information. It has been shown that sparse representations can be used to uncover meaningful and interpretable features as shown by the pen-stroke like features when learning sparse code for MNIST handwritten digits dataset, see Chapter 1 Section 7.1. It has been shown that brains use sparse activation patterns of neurons to represent information too (Ohki et al., 2005). Thus, if we want to solve the problems similar to what brains solve, like identifying similar things or patterns in a noisy environment while still being able to distinguish them, it seems sparse representations can be helpful. We showed in Chapter 1 that some of the properties of sparse representations lends it to represent similarity between things well. It is also very noise robust. We showed in the following chapters how sparse representations can used to solve problems in AI and Molecular Biology. Both the fields deal with very complex data that is prone to high amounts of noise. Therefore, sparse representations are well suited to simplify some problems in each of these areas.

In Chapter 2, it was shown how one of the methods of producing sparse representations, Hierarchical Temporal Memory (HTM) can be used to build a sensory memory for Learning Intelligent Decision Agent (LIDA) cognitive architecture. Sparse representations allow agents created from this cognitive architecture to create symbolic representations of patterns of input, while keeping the symbols grounded in the sensory apparatus of the agent. Using sparse representations, we were able to produce information rich representations for Vector LIDA

model. These can then be translated with perfect fidelity back and forth to symbolic representations that are better for knowledge representation and processing in a cognitive architecture.

Chapter 3 described a way of using sparse representations for gene expression data. A neural network based sparse autoencoder was trained on more than 100 thousand samples of publicly available gene expression samples on GEO Omnibus. We were able to show that the network learned biologically relevant features when compared between groups of samples with different conditions. The activation patterns of the hidden units also provide a signature for some conditions. This can be used in future to build a classifier that used this signature to do classification. This network will also help in discovering novel relationships between genes and conditions.

This shows that for many AI related applications it is worth exploring sparse representations as a possible method of representation of information.

2. Contributions

Various contributions made for the work described in this dissertation are listed below-

1. I proposed a method of building a sensory memory for LIDA cognitive architecture that allows us to create information rich representations in LIDA.
2. I proposed a method of grounding the vector representations used in Vector LIDA using an HTM based sensory memory.
3. I proposed a method to go back and forth with perfect fidelity between the two representations i. e. sparse representations and Modular Composite Representations.

4. The proposed methods enabled LIDA to have rich representations and allowed it to learn from these representations while still adhering to the Conscious Learning commitment.
5. I created an open-source, fast and scalable implementation of Integer Sparse Distributed Memory and Modular Composite Representations in Python.
6. I demonstrated a method of learning data-driven features in an unsupervised learning system using sparse representations from a trained sparse autoencoder.
7. I developed a method of inferring statistically significant and relevant biological gene sets in a set of gene expression samples with a common condition from a sparse autoencoder trained on gene expression data.
8. I was able to demonstrate that the network produces distinguishable activation patterns from groups of gene expression samples with different conditions.
9. I was able to show that the gene sets that correspond to statistically significant differentially active units between two groups of gene expression samples are also found to have a major overlap with statistically significant gene sets found using conventional methods of gene expression analysis.
10. I created an open-source implementation of a sparse autoencoder in Python with a TensorFlow backend. It allows a lot of features that can be activated as needed for training, like tied-weights, denoising, dropout etc. It also allows logging and serialization and deserialization of models.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., … Zheng, X. (2016). TensorFlow: A system for large-scale machine learning. *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 265–283. Retrieved from https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf

Aggleton, J. P., & Pearce, J. M. (2001). Neural systems underlying episodic memory: insights from animal research. *Philos Trans R Soc Lond B Biol Sci, 356*(1413), 1467-1482. doi: 10.1098/rstb.2001.0946

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., … Zheng, X. (2016). TensorFlow: A system for large-scale machine learning. *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 265–283. Retrieved from https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf

Ahmad, S., & Hawkins, J. (2015). Properties of sparse distributed representations and their application to hierarchical temporal memory. *ArXiv Preprint ArXiv:1503.07469*.

Ashburner, M., Ball, C. A., Blake, J. A., Botstein, D., Butler, H., Cherry, J. M., … others. (2000). Gene ontology: tool for the unification of biology. *Nature Genetics*, *25*(1), 25.

Berger, T. K., Perin, R., Silberberg, G., & Markram, H. (2009). Frequency-dependent disynaptic inhibition in the pyramidal network: a ubiquitous pathway in the developing rat neocortex. *The Journal of Physiology*, *587*(22), 5411–5425.

Bourlard, H., & Kamp, Y. (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, *59*(4–5), 291–294.

Candes, E., & Romberg, J. (2005). l1-magic: Recovery of sparse signals via convex programming. *URL: Www. Acm. Caltech. Edu/L1magic/Downloads/L1magic. Pdf*, *4*, 14.

Daigle Jr, B. J., Deng, A., McLaughlin, T., Cushman, S. W., Cam, M. C., Reaven, G., … Altman, R. B. (2010). Using pre-existing microarray datasets to increase experimental power: application to insulin resistance. *PLoS Computational Biology*, *6*(3), e1000718.

Dinsmore, J. (2014). *The symbolic and connectionist paradigms: closing the gap*. Psychology Press.

Donoho, D. L. (2006). Compressed sensing. *IEEE Transactions on Information Theory*, *52*(4), 1289–1306.

Fakoor, R., Ladhak, F., Nazi, A., & Huber, M. (2013). Using deep learning to enhance cancer diagnosis and classification. *Proceedings of the International Conference on Machine Learning*.

Franklin, S. (1997). *Artificial Minds* (1st ed.). Cambridge, MA, USA: MIT Press.

George, D., & Hawkins, J. (2005). A hierarchical Bayesian model of invariant pattern recognition in the visual cortex. *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference On*, *3*, 1812–1817. IEEE.

George, D., & Hawkins, J. (2009). Towards a mathematical theory of cortical micro-circuits. *PLoS Computational Biology*, *5*(10), e1000532.

Hawkins, J., Ahmad, S., & Dubinsky, D. (2011). Hierarchical Temporal Memory Including HTM Cortical Learning Algorithms, 0.2. Numenta. *Inc., September*.

Hofstadter, D. R., & Mitchell, M. (1994). *The Copycat project: A model of mental fluidity and analogy-making.*

Kanehisa, M., & Goto, S. (2000). KEGG: kyoto encyclopedia of genes and genomes. *Nucleic Acids Research*, *28*(1), 27–30.

Kanerva, P. (1988). *Sparse distributed memory*. MIT press.

Kanerva, P. (2009). Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive Computation*, *1*(2), 139–159.

Kawaguchi, Y., & Shindou, T. (1998). Noradrenergic excitation and inhibition of GABAergic cell types in rat frontal cortex. *Journal of Neuroscience*, *18*(17), 6963–6976.

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., & others. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

Lee, H., Battle, A., Raina, R., & Ng, A. Y. (2007). Efficient sparse coding algorithms. *Advances in Neural Information Processing Systems*, 801–808.

Liberzon, A., Subramanian, A., Pinchback, R., Thorvaldsdóttir, H., Tamayo, P., & Mesirov, J. P. (2011). Molecular signatures database (MSigDB) 3.0. *Bioinformatics*, *27*(12), 1739–1740. https://doi.org/10.1093/bioinformatics/btr260

Liou, C.-Y., Cheng, W.-C., Liou, J.-W., & Liou, D.-R. (2014). Autoencoder for words. *Neurocomputing*, *139*, 84–96.

Mills, K. I., Kohlmann, A., Williams, P. M., Wieczorek, L., Liu, W., Li, R., … Haferlach, T. (2009). Microarray-based classifiers and prognosis models identify subgroups with distinct clinical outcomes and high risk of AML transformation of myelodysplastic syndrome. *Blood*, *114*(5), 1063–1072. https://doi.org/10.1182/blood-2008-10-187203

Minsky, M. (1974). *A framework for representing knowledge*.

Ng, A., & others. (2011). Sparse autoencoder. *CS294A Lecture Notes*, *72*(2011), 1–19.

Ohki, K., Chung, S., Ch'ng, Y. H., Kara, P., & Reid, R. C. (2005). Functional imaging with cellular resolution reveals precise micro-architecture in visual cortex. *Nature*, *433*(7026), 597.

Olshausen, B. A., & Field, D. J. (1996). Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, *381*(6583), 607.

Olshausen, B. A., & Field, D. J. (1997). Sparse coding with an overcomplete basis set: A strategy employed by V1? *Vision Research*, *37*(23), 3311–3325.

Olshausen, B. A., & Field, D. J. (2004). Sparse coding of sensory inputs. *Current Opinion in Neurobiology*, *14*(4), 481–487.

Poultney, C., Chopra, S., Cun, Y. L., & others. (2007). Efficient learning of sparse representations with an energy-based model. *Advances in Neural Information Processing Systems*, 1137–1144.

Rinkus, G. J. (1995). *TEMECOR: An associative, spatio-temporal pattern memory for complex state sequences*.

Rinkus, G. J. (2004). A neural model of episodic and semantic spatiotemporal memory. *Proceedings of the Annual Meeting of the Cognitive Science Society, 26*.

Rinkus, G. J. (2014). Sparsey™: event recognition via deep hierarchical sparse distributed codes. *Frontiers in Computational Neuroscience, 8*, 160.

Ritchie, M. E., Phipson, B., Wu, D., Hu, Y., Law, C. W., Shi, W., & Smyth, G. K. (2015). limma powers differential expression analyses for RNA-sequencing and microarray studies. *Nucleic Acids Research, 43*(7), e47–e47.

Sabates-Bellver, J., Van der Flier, L. G., de Palo, M., Cattaneo, E., Maake, C., Rehrauer, H., … Marra, G. (2007). Transcriptome profile of human colorectal adenomas. *Molecular Cancer Research: MCR*, *5*(12), 1263–1275. https://doi.org/10.1158/1541-7786.MCR-07-0267

Smolensky, P. (1987). Connectionist AI, symbolic AI, and the brain. *Artificial Intelligence Review*, *1*(2), 95–109.

Steinert, R., Rehn, M., & Lansner, A. (2006). Recognition of handwritten digits using sparse codes generated by local feature extraction methods. *ESANN*, 161–166.

Subramanian, A., Narayan, R., Corsello, S. M., Peck, D. D., Natoli, T. E., Lu, X., … Golub, T. R. (2017). A Next Generation Connectivity Map: L1000 Platform and the First 1,000,000 Profiles. *Cell*, *171*(6), 1437-1452.e17. https://doi.org/10.1016/j.cell.2017.10.049

Tan, J., Doing, G., Lewis, K. A., Price, C. E., Chen, K. M., Cady, K. C., … Greene, C. S. (2017). Unsupervised extraction of stable expression signatures from public compendia with an ensemble of neural networks. *Cell Systems*, *5*(1), 63–71.

Tan, J., Huyck, M., Hu, D., Zelaya, R. A., Hogan, D. A., & Greene, C. S. (2017). ADAGE signature analysis: differential expression analysis with data-defined gene sets. *BMC Bioinformatics*, *18*(1), 512. https://doi.org/10.1186/s12859-017-1905-4

Taroni, J. N., Grayson, P. C., Hu, Q., Eddy, S., Kretzler, M., Merkel, P. A., & Greene, C. S. (2018). *MultiPLIER: a transfer learning framework reveals systemic features of rare autoimmune disease*.

Väremo, L., Nielsen, J., & Nookaew, I. (2013). Enriching the gene set analysis of genome-wide data by incorporating directionality of gene expression and combining statistical hypotheses and methods. *Nucleic Acids Research*, *41*(8), 4378–4391. https://doi.org/10.1093/nar/gkt111

Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. *Proceedings of the 25th International Conference on Machine Learning*, 1096–1103. ACM.

Winston, P. H. (1992). *Artifical Intelligence. 3-rd Ed.* Addison Wesley.

Xu, J., Xiang, L., Liu, Q., Gilmore, H., Wu, J., Tang, J., & Madabhushi, A. (2016). Stacked sparse autoencoder (SSAE) for nuclei detection on breast cancer histopathology images. *IEEE Transactions on Medical Imaging*, *35*(1), 119–130.

Yu, G., Li, F., Qin, Y., Bo, X., Wu, Y., & Wang, S. (2010). GOSemSim: an R package for measuring semantic similarity among GO terms and gene products. *Bioinformatics*, *26*(7), 976–978. https://doi.org/10.1093/bioinformatics/btq064

Zhang, Z., Xu, Y., Yang, J., Li, X., & Zhang, D. (2015). A survey of sparse representation: algorithms and applications. *IEEE Access*, *3*, 490–530.