A logical re-conception of neural networks:

Hamiltonian bitwise part-whole architecture

An Undergraduate Thesis in Computer Science

Clay Foye

Advisors: Richard Granger, Ph.D;

Soroush Vosoughi, Ph.D

June 1, 2022

Contents

1 Abstract

Contemporary neural networks such as Convolutional Neural Networks (CNN)'s rely on a constrained set of linear-algebra premises. We describe a novel Hamiltonian Logic Network (Hnet) which derives from simple principles of binary logic operations, with an aim of working toward an architecture of neuro-symbolic AI. The Hnet performs classification and recognition and constructs part-whole relations and hierarchies using the intrinsically low-cost architecture of binary nodes and bit-wise arithmetic operations. By compositing these relationships using Hamiltonian hierarchies, we are able to create a sparse, non-adjacent tiling of any space. The Hnet's capabilities are demonstrated on two classical datasets: the MNIST characters dataset and UCI's German Credit dataset.

2 Introduction

Deep learning is a subset of machine learning which seeks to address the shortcomings of the field in high-dimensional space [3]. Deep learning refers to the use of multi-layer (deep) neural networks. The layers of a deep neural network consist of teachable "nodes [...] many of which compute non-linear input-output mappings" [10]. Each node can be considered as its own linear regression model with individual weights and threshold values [14]. A node's output value is determined by the following pair of equations [14]:

$$(1) \qquad \sum_{i=1}^{m} w_i x_i + \text{bias} = w_1 x_1 + w_2 x_2 + w_3 x_3 + ... + \text{bias}$$

$$(2) \qquad \text{output} = \begin{cases} 1 & \text{if} \sum_{i=1}^{m} w_i x_i + b \geq 0 \\ 0 & \text{if} \sum_{i=1}^{m} w_i x_i < 0 \end{cases}$$

A node's activation, therefore, depends heavily on its weights $\vec{w}$ for a given input vector $\vec{x}$. Weights almost always take the form of extremely precise floating-point numbers. Their precision allows deep neural networks to endlessly tweak and adjust themselves.

Deep neural networks are able to handle large amounts of raw input data, while traditional "shallow" networks — networks with at most a few layers — require preprocessing work to be done by humans. Deep neural networks also perform "end-to-end" learning; beginning with just raw data and a goal, they are able to figure out for themselves how to complete their task with the data given [24]. While shallow networks plateau the more data is provided to the network, deep networks can scale endlessly [24]. Deep neural networks therefore proved to be the forefront of machine learning's attempt to find structure in high-dimensional data [10].

2.1 Convolutional Neural Networks

One of the most popular types of deep neural networks is the Convolutional Neural Network (CNN), which specialize in classification and image recognition tasks [2]. Some examples include identifying handwritten numbers [4, 11] and large-scale image classification [9, 18, 21, 6].

CNN's are made up of three types of layers: convolutional layers, pooling layers, and a fully-connected layer [2]. Convolutional layers rely on feature detectors which compute weighted the dot products of filter values against pixel values of an image. In addition to typical gradient descent, these weights are updated by layers further along in the network in a process known as back-propagation [11]. Before passing their values to the next layer, convolutional layers often run their outputs through a ReLu activation function (Equation 3) to introduce non-linearity. [2, 10]

$$x = max(x, 0) \tag{3}$$

Pooling layers of a CNN perform dimensionality reduction from the outputs of the convolutional layers. These pooling layers allow CNN's to combine low-level features into high-level features [10]. The reduced outputs of the pooling layers are then passed to the fully-connected layer, a typical deep neural network structure which uses a softmax activation function to classify images or identify features [2].

2.2 Recurrent Neural Networks

While CNN's are excellent at performing classification and image-feature recognition, Recurrent Neural Networks (RNN's) specialize in sequential data, such as speech and language recognition [16, 10]. RNN's use input data which is

given in a sequence, incorporating each datapoint (such as a word or character) into its own layer. The high-cost process of gradient descent paired with backpropagation allows RNN's to become the foremost model for many different tasks. They are able to predict the next value in a sequence: such as a letter, word, or even a paragraph [20, 13]. Training an RNN on multiple languages has created advances in translation [1]. RNN's can even start to "memorize" complicated narratives and answer questions about them [5, 22, 23].

2.3 Neuro-Symbolic AI

CNN's and RNN's have begun to approach "comprehension," a complicated understanding of data and the ability to answer questions about it. However, current neural networks rely on endless amounts of data and processing power to achieve a semblance of human-style comprehension [15]. Neuro-symbolic networks attempt to re-think the high-precision architecture of traditional deep neural networks in favor of "symbolic representations of problems and logic" [15]. Instead of focusing on a single task, neuro-symbolic architecture attempts to create a foundation of basic rules or relations. It then builds this basic architecture into complex representations and relations. Thus, neuro-symbolic AI places emphasis on part-whole relations and tree-like parent-child relationships [12, 19]. These relationships greatly reduce the need for large amounts of data and processing power.

An early attempt at this kind of architecture was Geoffrey E. Hinton et al's capsules [17]. Capsules are a "group of neurons" with a corresponding "activity vector" [17]. Activity vectors store "instantiation parameters" for an "object or an object part" [17]. Thus, the part-whole relationships of objects are embedded in capsule architecture. These capsules are organized in a hierarchy: layers of the deep neural network are linked through parent-child relationships

between capsules. This architecture creates the desired part-whole structure using a "routing-by-agreement" mechanism. The routing-by-agreement mechanism makes it such that a capsule prefers to "send its output to higher-level capsules whose activity vectors have a high scalar product with the prediction coming from the lower-level capsule." This model still relies on scalar products and other linear algebra principles of neural networks. The architecture ties directly to neuro-symbolic architecture by creating hierarchical relationships between capsules. Thus, symbolic representations of objects and object parts are incorporated using the architecture of the model itself.

The Compositional Language and Elementary Visual Reasoning (CLEVR) dataset is a recent landmark in the field of neuro-symbolic AI [8]. The CLEVR dataset contains typical reasoning questions about images filled with different colored shapes [8]. However, the CLEVR dataset also included "spatial reasoning" questions, which "forced models to reason about objects' relative positions." Lacking the appropriate architecture, modern neural networks' performance showed a "significant drop." Many different neuro-symbolic models have risen to meet the CLEVR or a CLEVR-like challenge [25, 7]. These models each try to create an architecture that relies on part-whole relations which are able to build upon themselves. A few years later, the CoLlision Events for Video REpresentation and Reasoning (CLEVRER) dataset was introduced [26]. It contains videos of collisions between different colored shapes, and asks descriptive, explanatory, predictive, and counterfactual questions. CLEVRER introduced the temporal to the CLEVR dataset, forcing models to comprehend temporal reasoning in addition to spatial reasoning. These datasets were important because they revealed the shortcomings of traditional deep neural networks and paved the way for neuro-symbolic AI.

2.4 Motivation

Inspired by neuro-symbolic AI's part-whole approach, we have designed an architecture which creates these part-whole relations using logic operations encoded and composited using Hamiltonian matrices. The Hnet architecture abandons the high-precision operations of previous neuro-symbolic networks in favor of binary logic, relying on inherently low-precision operations. We intended to create a model which is able to comprehend complex relationships by building upon simple binary relationships.

3 Processing Input Data

Input data is always understood by Hnet to be a directed graph. Hnet's data processing is most easily explained in the case of explicitly binary node data. We will then extend our explanation to data which is not explicitly binary.

We begin with a binarized version of the MNIST characters dataset; a black pixel is interpreted as a 1, while a white pixel as 0. Node adjacencies are the pixels adjacent in each cardinal direction. All edges, by convention, lead from right-to-left or top-to-bottom. Thus, in Hnet data, there is always a single sink reachable from all nodes and a single source unreachable from any other node. After constructing the graph, each edge must consist of one of four different simple binary logical relationships: NOR (0,0), NCONV (0,1), NIMPL (1,0), or AND (1,1). These binary values can also be interpreted as boolean values (T, F). This graph of edges is our input graph for an image, ready to be trained or tested on by Hnet.

Other datasets, such as the UCI's German Credit dataset, consist of multiple fields of categorical and continuous data. Again we binarize for simplicity; in this case by binning the continuous data into categories, and then convert all categorical data using one-hot encodings to binary data. We then build a directed graph using the one-hot values of each possible attribute as nodes. Following the Hnet graph structure, there is a single source node unreachable from any other nodes, and a single sink node reachable from every other node. It is in this manner that we create a graph made up of our four logical relations.

4 Training

After processing our data into input graphs, we are ready to begin training the Hnet. In order to select training and testing data, we randomly split the data in half.

Building Hamiltonians is the key for Hnets to learn data and compare them against components found in testing data. After building a Hamiltonian for an input graph found in training data, the Hamiltonian is tagged with the respective class of the input data. This information is later used in the testing stage to identify data points to trained classes.

In order to build a Hamiltonian matrix, we create a square matrix whose rows and columns are the number of possible values. For our two datasets, that meant the total number of pixels or the total number of possible attributes for the credit dataset. Before we build a Hamiltonian matrix using these components, we need to derive the Hamiltonian matrices of each binary relation, and describe the process by which we composite these Hamiltonians. Using the process described above for our large Hamiltonian matrices, we can derive the simple initial binary Hamiltonian matrices.

4.1 Deriving the Four Basic Hamiltonians

Applying a Hamiltonian matrix to a test datum yields an energy value E whose best values are those that are lowest. Energy is calculated using the following equation:

$$(4) \qquad E_r = \begin{bmatrix} x & y \end{bmatrix} \hat{H}_r \begin{bmatrix} x \\ y \end{bmatrix} + k_r$$

Where E_r is Energy, the vector $\begin{bmatrix} x & y \end{bmatrix}$ is the test data point, $\hat{H}_r$ is the stored Hamiltonian, and k_r is some constant associated with $\hat{H}_r$ which ensures that E is non-negative. We expand $\hat{H}_r$ to a 4x4 matrix, which corresponds to the Hamiltonian of a binary logic relation.

$$(5) \qquad E_r = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} a & b \\ b & c \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + k_r$$

Thus:

$$(6) \qquad E_r = ax^2 + 2bxy + cy^2 + k_r$$

for a binary logic relation. We can solve this equation for a, b and c, for the four binary pairs, shown in Figure 1.

s		E(s)			
x	y	E(0,0)	E(0,1)	E(1,0)	E(1,1)
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

Figure 1: The four binary relations and their corresponding energies.

We are thus able to derive the following Hamiltonians for these pairs. The derived Hamiltonians are shown in Figure 2.

$$\hat{H}_{AND} = \begin{bmatrix} 0 & -1/2 \\ -1/2 & 0 \end{bmatrix} \quad \hat{H}_{NCONV} = \begin{bmatrix} 0 & 1/2 \\ 1/2 & -1 \end{bmatrix}$$

$$\hat{H}_{NOR} = \begin{bmatrix} 1 & -1/2 \\ -1/2 & 1 \end{bmatrix} \quad \hat{H}_{NIMPL} = \begin{bmatrix} -1 & 1/2 \\ 1/2 & 0 \end{bmatrix}$$

Figure 2: The four basic Hamiltonians.

Observe that these Hamiltonians also conform to the relative relationships

of their corresponding logic operations. For example: $NAND = \neg(AND) \implies \hat{H}_{NAND} = -\hat{H}_{AND}$.

4.2 Compositing Hamiltonians

In order to construct Hamiltonians of larger sizes, such as the size of components, we composite these basic components into larger ones. Using the basic binary relations and their derived Hamiltonians, we can composite any dimensional Hamiltonian matrix. We may do this by projecting each basic H into the 2-dimensional subspace of our new composite Hamiltonian. Thus, the resulting Hamiltonian is:

$$(7) \qquad H = \sum_{(u,v,r,)\in R} H_{u,v,r} + k_r$$

We will demonstrate using the graph α with four nodes n_1, n_2, n_3, n_4, and the directed edges e_1, e_2, e_3, e_4:

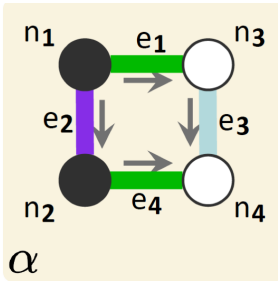

Figure 3: Graph α. Edge colors correspond to relations: green corresponds to NIMPL, purple to AND, and cyan to NOR.

Thus, we are able to create a Hamiltonian of dimension 4x4, as demonstrated in Figure 4. As stated earlier, the rows and columns of a composite Hamiltonian correspond to the number of edges in an input graph. Figure 6 shows our completed Hamiltonian for Graph Alpha.

In this manner, we are able to composite Hamiltonians for any input graph.

$$\hat{H}_{e_1} = \begin{pmatrix} -1 & 1/2 \\ 1/2 & 0 \end{pmatrix} \quad \hat{H}_{e_2} = \begin{pmatrix} 0 & -1/2 \\ -1/2 & 0 \end{pmatrix}$$

$$\hat{H}_{e_3} = \begin{pmatrix} 1 & -1/2 \\ -1/2 & 1 \end{pmatrix} \quad \hat{H}_{e_4} = \begin{pmatrix} -1 & 1/2 \\ 1/2 & 0 \end{pmatrix}$$

$$\hat{H}_{\alpha} = \begin{pmatrix} -1+0 & -1/2 & 1/2 & \\ -1/2 & 0-1 & & 1/2 \\ 1/2 & & 0+1 & -1/2 \\ & 1/2 & -1/2 & 1+0 \end{pmatrix}$$

$$\hat{H}_{\alpha} = \begin{pmatrix} -1 & -1/2 & 1/2 & 0 \\ -1/2 & -1 & 0 & 1/2 \\ 1/2 & 0 & 1 & -1/2 \\ 0 & 1/2 & -1/2 & 1 \end{pmatrix}$$

Figure 4: A demonstration in compositing a Hamiltonian for graph α. The basic Hamiltonians of each edge of graph α are labeled and given. Next, the process of compositing these Hamiltonians is demonstrated. Finally, the Hamiltonian of graph α is given.

5 Components

For each MNIST input graph, we find all connected components of length greater than 4 and less than 25. For each connected component, we composite a Hamiltonian using the process described above. Each connected component is assigned a class histogram, whose values correspond to the component's occurrence in each class. We call the resulting collection of components a "component bank."

We follow a more general process for the credit dataset. Using the input graphs of all datapoints, we unsupervisedly cluster the edges into components based upon their co-occurence in the same datapoint. So far, we have used the cluster methods of K-means and ICA.

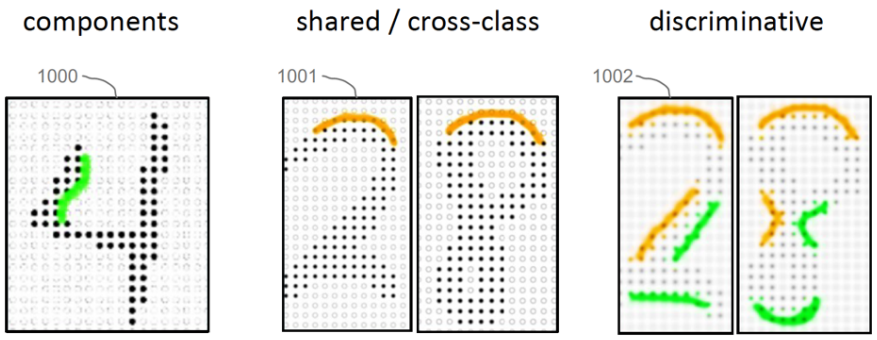

Figure 5: Different types of components, shown using images from the MNIST character dataset.

We refer to the components which are created out of the connected components of the input graphs as "Tier 1" components. Some Tier 1 components are shared across classes, such as the curve at the top of a two and an an eight. Combining the observed shared components allows Hnets to discriminate between classes using logic relationships. A particular connected component might exist in both a two and an eight, but the combination of observed connected components can only exist in a particular class, i.e. a two. Thus, the

inclusion of Tier 1 components to our testing process allows us to improve our discriminatory ability. It also allows us to build upon the framework of Tier 1 components. Building upon this framework allows us to create increasingly complex and removed part-whole relationships.

5.1 Energy Vectors

After calculating the Hamiltonians for every observed component, we are able to create energy vectors for each test image. An energy vector is of length n, where n is the number of observed components. Each value in a training datapoint's energy vector is the energy of the training data vector against the i_{th} component's Hamiltonian. Energy is calculated using equation (5).

6 Testing

Testing takes advantage of the composite Hamiltonians of the connected components which we learned during our testing phase, as well as the stored energy vectors of the training data. In order to use these, we first create input graphs from our testing data. We then create energy vectors for every testing datapoint in the same way as our training data. We then find the K-nearest-neighbor training datapoint energy vector. Finally, we identify the training datapoint with the class of the nearest training neighbor.

7 Permutations

Permutations are our method of recognizing the same connected component in a general area of a photograph. This method currently has only been applied to the MNIST characters dataset. First, we assign all of our trained components to their own group. Then, for each connected component, we apply affine transformations to each component's Hamiltonian. The transform of a given Hamiltonian of a connected component is:

$$\hat{H}' = P^T \hat{H} P \tag{8}$$

Where P is the permutation operator. For the MNIST character dataset, these transformations correspond to translations of a given part by a set of fixed distances (1, 2, or 3) pixels in the four cardinal directions. After permuting the Hamiltonians, we store the results in the group which the Hamiltonian originated in. During testing, we assign the energy of a specific group to be the minimum energy of any Hamiltonian in that group computed against a test image.

8 Meta-Components

Tier 1 components can be organized into binary relations with each other, forming what we call "Meta Components." Combining components can be done ad infinitum, forming more complex symbolic representations of the data. This process corresponds to our goal of creating a neuro-symbolic like architecture which is able to composite basic rules for complex comprehension. The method by which we place Tier 1 components in relation to each other differs from dataset to dataset. In both cases, we create a sparse tiling of the input space.

For MNIST characters, we use the visual nature of the images to create Meta Components. We take advantage of this by using the spatial relationships of connected components to create a 9x9 square graph which we are able to composite into a Hamiltonian. We then use these meta-relationships of the components in order to create the energy vectors for our training and testing images.

Meanwhile, for the credit dataset, we create a fully-connected graph of Tier-1 components found in each training datapoint. Activation of a node in each graph is determined by an energy threshold of each Tier 1 Hamiltonian computed against its respective datapoint. We then unsupervisedly cluster these edges into n different Meta Components using K-means or ICA. As in the MNIST dataset, we then use these Meta Components to create the energy vectors for our training and testing images.

9 Sample Results

Included in this section are actual outputs from sample runs on each dataset.

9.1 MNIST Outputs

We described how Tier 1 components are constructed in the section **Components**. Tier 1 components in the MNIST dataset are made up of actual edges from the source images. The goal of our model was to create a sufficient basic structure and the ability to composite this structure into more complex representations of data. Our basic structure of binary logic relations was able to build upon its own simple rules to create these complex relations. The components shown below are examples of this part-whole architecture. Individual pixels and edges are visible in Figures 6 and 7. These edges have been grouped together into these components, forming a parent-child relationship typical of neuro-symbolic architecture.

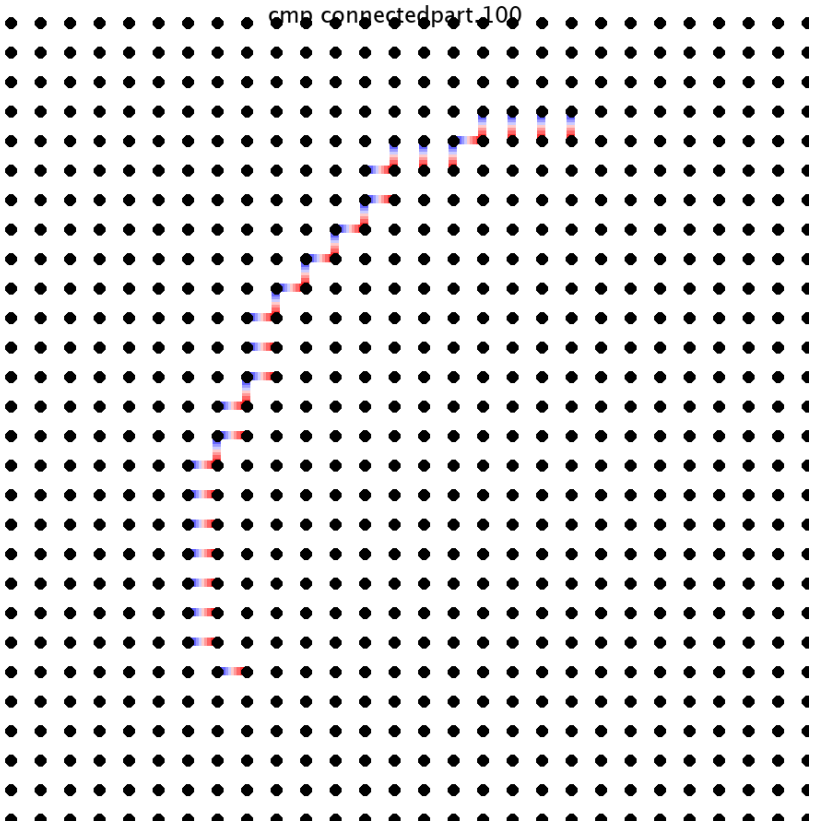

Figure 6: The displayed edges correspond to the relations for a pair of connected pixels. In this figure, we indicate the pixel values using directed edges. The blue side of an edge corresponds to a pixel value of 0, while the red side corresponds to a pixel value of 1.

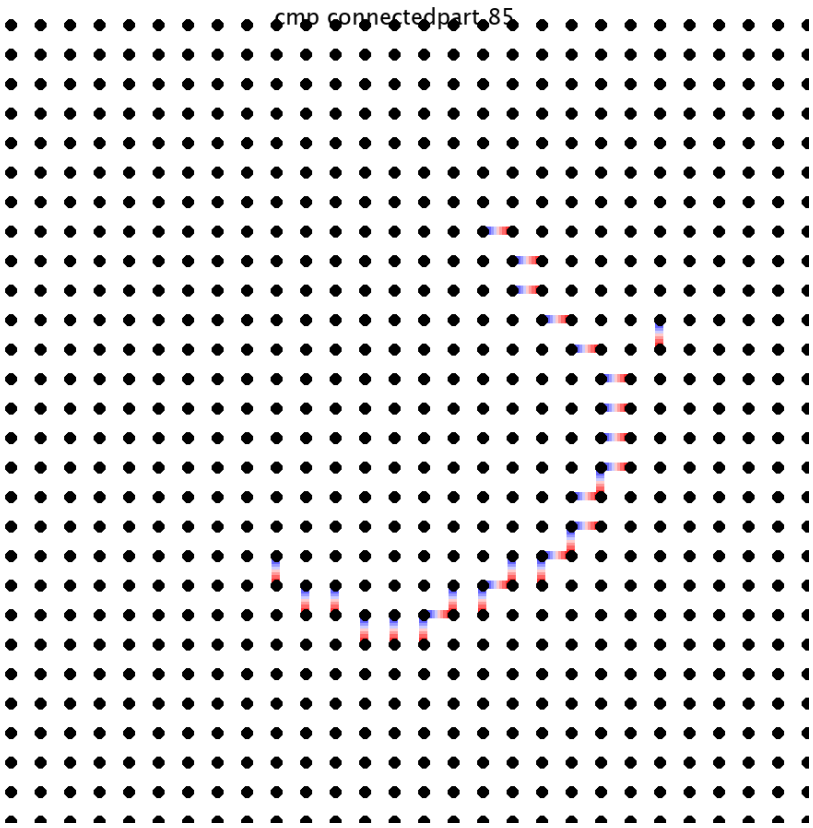

Figure 7: The displayed edges correspond to the relations where for a pair of connected pixels. See Figure 1 for further description.

During testing and training, we are able to compute class histograms for all Tier 1 components.

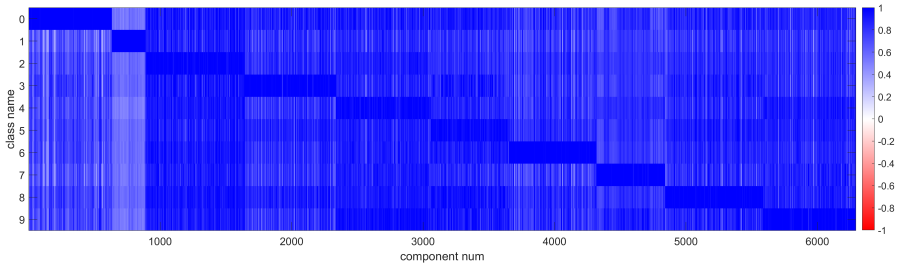

Figure 8: On the horizontal axis of the histogram is the component ID or "num." On the vertical axis is the class label (the characters 0-9). The class histogram is a heatmap of each component's predictability for every class. The hotter the section, the more predictive the corresponding component for the corresponding class. This is the class histogram of Tier 1 components against training data.

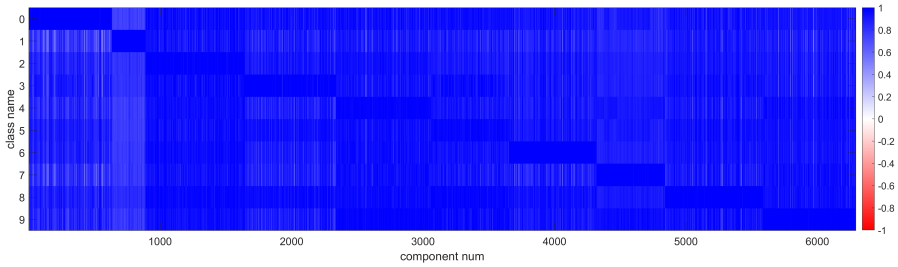

Figure 9: The class histogram for Tier 1 components against testing data. See Figure 8 for more information on class histograms.

9.2 UCI Credit Outputs

In the **Components** section, we described how Tier 1 components are constructed out of the UCI Credit dataset. Because the dataset is not inherently pictoral, we record these components using what we call "circle diagrams." Our entire architecture is dependant upon logic relations. Therefore, these components are a natural composition of these relations. Following in the part-whole structure of neuro-symbolic AI, our architecture builds upon basic symbolic representations to create complex ones. These circle diagrams are demonstrations of how these basic logic relationships can be composited to comprehend data.

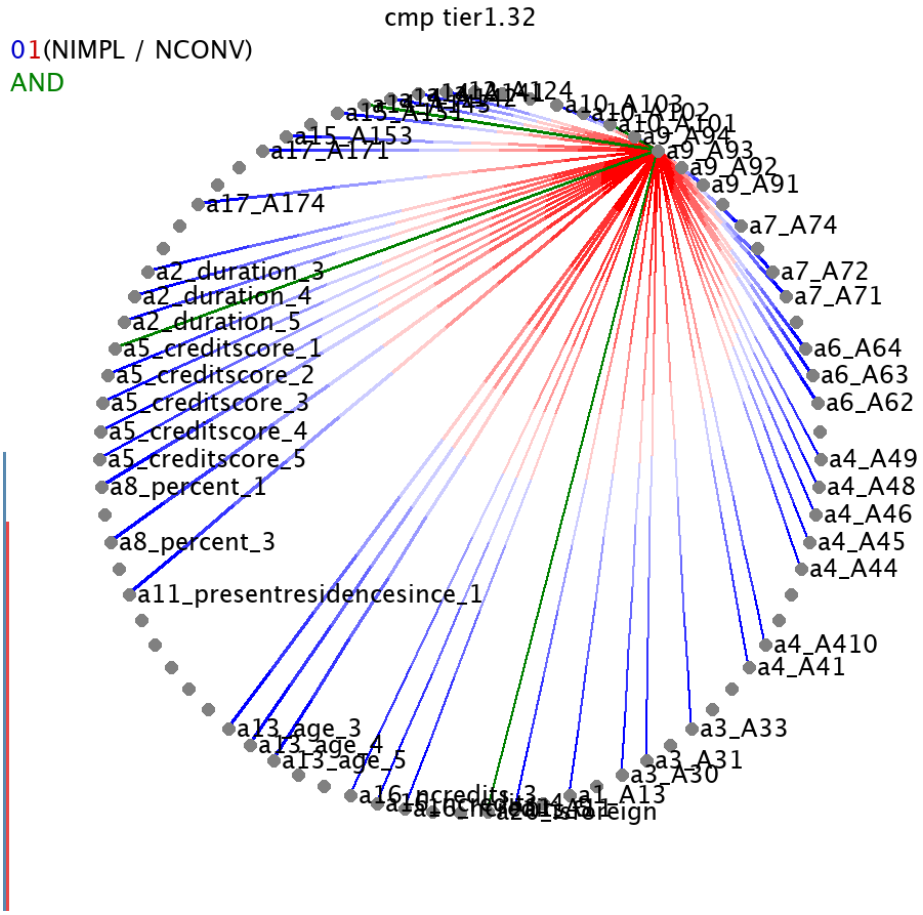

Figure 10: A Circle Diagram for Component 32. Each node on Tier 1 circle diagram corresponds to a possible binned category. Nodes with non-zero degree have their labels printed. These labels are shorthand provided by the dataset. As in the Tier 1 components of the MNIST dataset, the edges are directed, with heat describing direction. Some components contain nodes connected by a green AND. These connections are between two activated nodes.

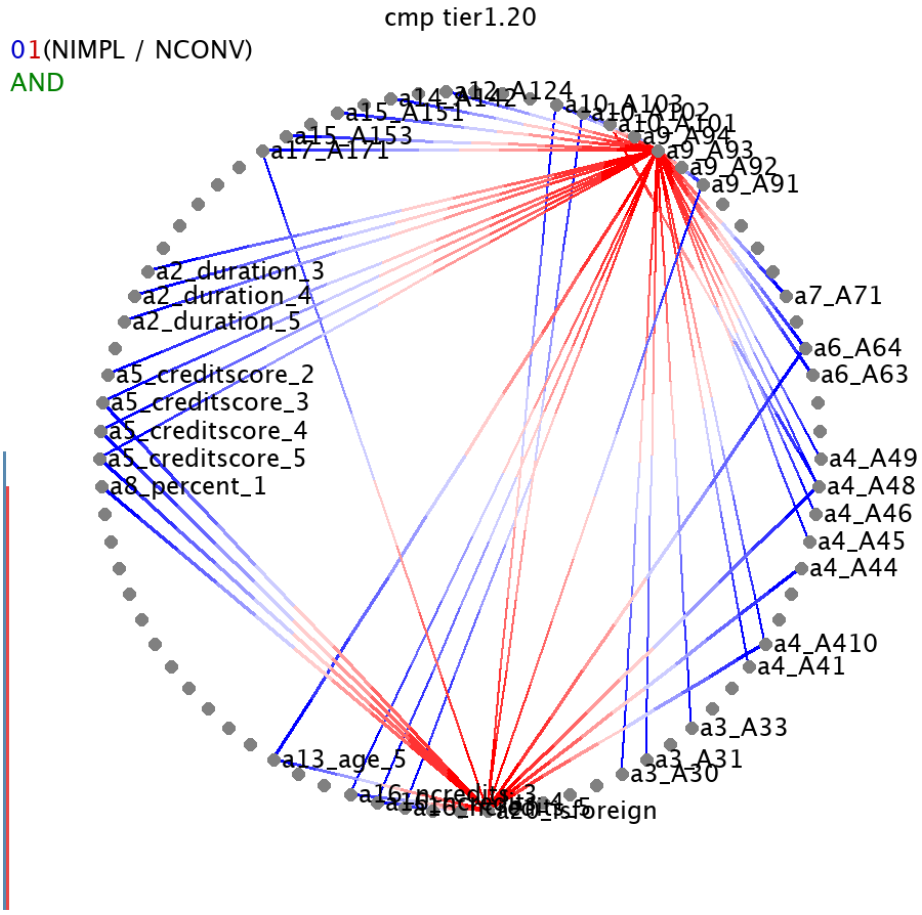

Figure 11: Another example of a Tier 1 circle diagram.

As with the MNIST dataset, we are able to create class histograms for Tier 1 components from the UCI credit dataset. As in the MNIST class histograms, the component ID is given on the horizontal axis, while the class label is given on the vertical axis. The hotter the section, the more predictive the corresponding component for the corresponding class.

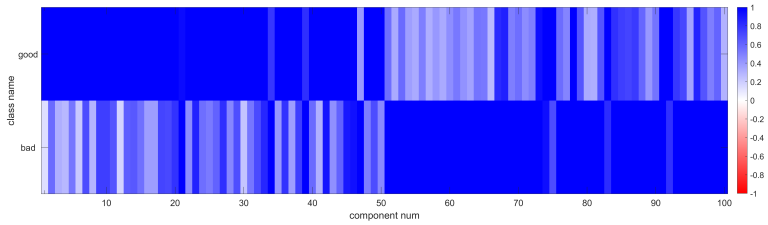

Figure 12: A class histogram for Tier 1 components on the training data. See Figure 8 for more information on class histograms.

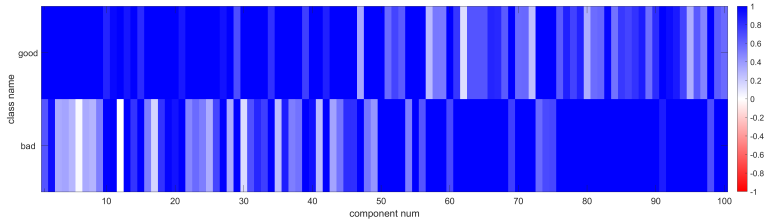

Figure 13: A class histogram for Tier 1 components on the testing data. See Figure 8 for more information on class histograms.

9.2.1 Meta Components

We described how Meta Components were constructed in the section **Meta Components**. Just as our foundational logic relations were composited into Tier 1 components, we composite those Tier 1 components into Meta Components. We preserve the low-cost hierarchical structure of our architecture, but increase the complexity of our model's comprehension of the data. These symbolic and literal parent-child and part-whole relationships are directly a result of our base architecture. This structure is aimed towards a neuro-symbolic understanding of machine learning.

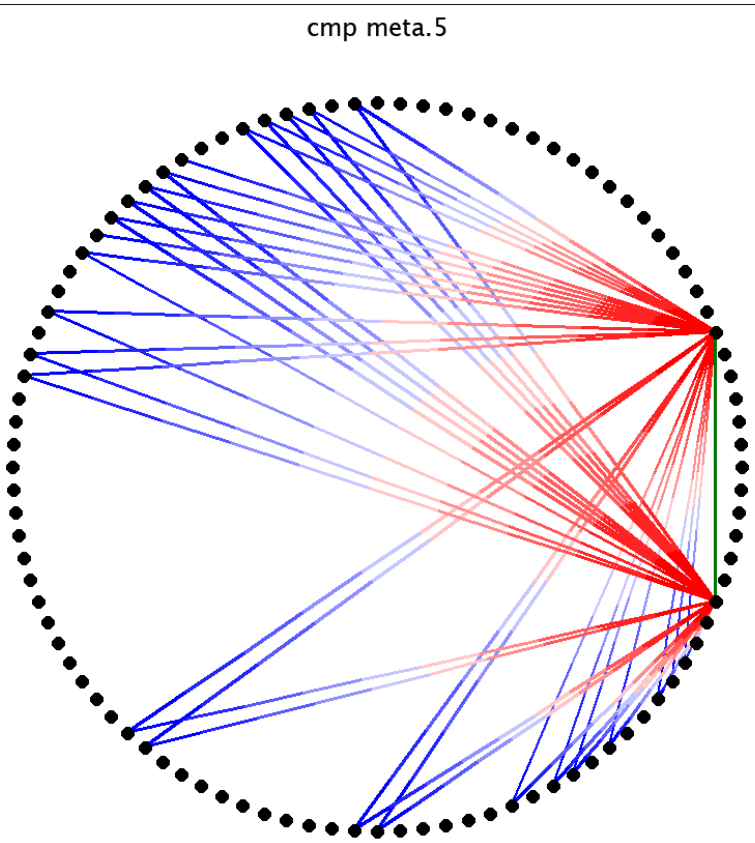

Figure 14: A diagram of Meta Component 5. We also use circle diagrams to show Meta Components. However, the nodes in the circle diagram of a Meta Component represent Tier 1 components, instead of possible attributes.

We also were able to create class histograms for the Meta Components. They follow the structure of the previous class histograms.

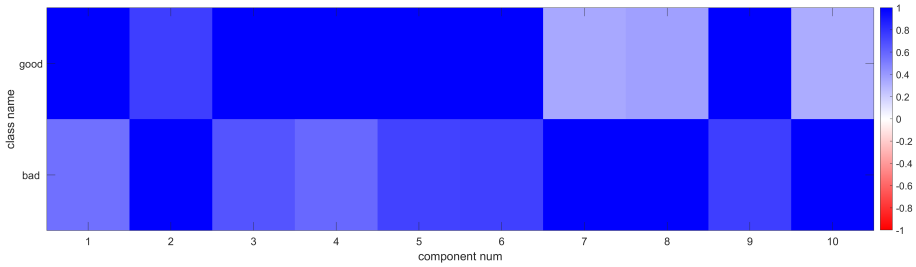

Figure 15: A class histogram for Meta Components on the training data. See Figure 8 for more information on class histograms.

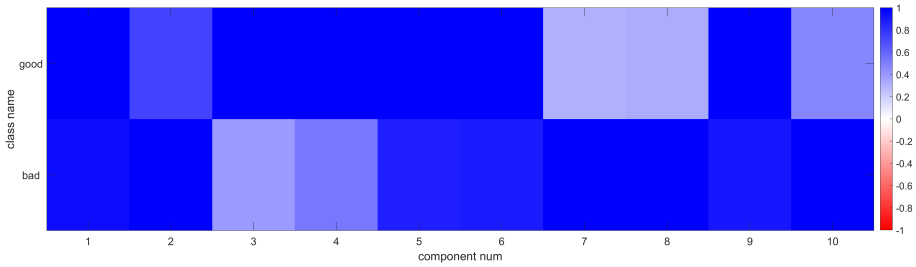

Figure 16: A class histogram for Meta components on the testing data. See Figure 8 for more information on class histograms.

10 Performance

We want to stress that it is not our intention to join the current mad rush for accuracy percentages. Instead, our goal is to present a neural network structure that offers a novel approach for the problems that current neural networks do not readily handle. Rather than as a substitute, the presented Hnet method could be used as an adjunct to other ANN methods, such that the Hnet provides enriched symbolic representations that otherwise would not be identified.

Presented in this section are the results of several tests. In the following tables, we show accuracy on testing and training datasets. The first two tables show results for the UCI Credit dataset. The second two tables each show a run on the MNIST characters dataset.

In the leftmost "analysis" column, the table lists which combination of methods was run. Runs which are labeled with "metacred" created Meta Components out of the UCI credit dataset and created energy vectors using them. Runs with the label "basiccred" only constructed Tier 1 components and created energy vectors using those. The second section in an "analysis" title gives the clustering method for the Tier 1 components: k-means, ICA, or ICA where many components are cropped. The third section gives the clustering method for Meta Components. If a run title only consists of "basiccred," then the run used Tier 1 components cropped extremely heavily by ICA. For the runs on the MNIST dataset, both runs use Tier 1 components that have been cropped down by ICA to only 100 total components. These runs are titled "basicimginit-connected.25."

The columns "tstacc" and "trnacc" give the accuracies for Hnet on the testing and training datasets, respectively. The following columns should be considered benchmarks. The columns "pixeltstacc" and "pixeltrnacc" give accuracy using simply the raw binarized data, as if we did not use an Hnet at all. Finally,

the columns “edgetstacc” and “edgetrnacc” is the accuracy if we used the edges amongst features to create energy vectors instead of the components the edges form.

analysis	tst_acc	trn_acc	pixel_tst_acc	pixel_trn_acc	edge_tst_acc	edge_trn_acc	timestamp
metacred_init-clust.tier1.ica.100.50-initmeta.10-clust.meta.kmeans.10.50	0.6	0.6	0.5967	0.5933	0.6467	0.5933	13-May-2022 15:10:05
metacred_init-clust.tier1.kmeans.100.50-initmeta.10-clust.meta.kmeans.10.50	0.5767	0.5933	0.5967	0.5933	0.6467	0.5933	13-May-2022 14:58:22
metacred_init-clust.tier1.kmeans.20.50-initmeta.10-clust.meta.kmeans.10.50	NaN	NaN	0.5967	0.5933	0.6467	0.5933	13-May-2022 15:06:14
metacred_init-clust.tier1.ica.100.50-initmeta.10-clust.meta.ica.10.50	0.5467	0.5233	0.5967	0.5933	0.6467	0.5933	13-May-2022 15:20:42
metacred_init-clust.tier1.icacrop.100.50-initmeta.10-clust.meta.kmeans.10.50	0.56	0.57	0.5967	0.5933	0.6467	0.5933	13-May-2022 15:31:07
metacred_init-clust.tier1.icacrop.100.50-initmeta.10-clust.meta.icacrop.10.50	0.56	0.6333	0.5967	0.5933	0.6467	0.5933	13-May-2022 15:41:27
metacred_init-clust.tier1.icacroplots.100.50-initmeta.10-clust.meta.kmeans.10.50	0.6567	0.6367	0.5967	0.5933	0.6467	0.5933	13-May-2022 15:52:11
metacred_init-clust.tier1.icacroplots.100.50-initmeta.10-clust.meta.icacroplots.10.50	0.6433	0.63	0.5967	0.5933	0.6467	0.5933	13-May-2022 16:03:20
metacred_init-clust.tier1.icacropsome.100.50-initmeta.10-clust.meta.kmeans.10.50	0.5667	0.5567	0.5967	0.5933	0.6467	0.5933	13-May-2022 16:14:38
metacred_init-clust.tier1.icacropsome.100.50-initmeta.10-clust.meta.icacropsome.10.50	0.6	0.62	0.5967	0.5933	0.6467	0.5933	13-May-2022 16:25:41
basiccred_init	0.6867	0.5767	0.5967	0.5933	0.6467	0.5933	13-May-2022 16:36:49
basiccred_init-clust.tier1.kmeans.100.Inf	0.7	0.6333	0.5967	0.5933	0.6467	0.5933	13-May-2022 16:52:44
basiccred_init-clust.tier1.kmeans.100.50	0.6567	0.6167	0.5967	0.5933	0.6467	0.5933	13-May-2022 17:01:29
basiccred_init-clust.tier1.kmeans.20.Inf	0.6633	0.6533	0.5967	0.5933	0.6467	0.5933	13-May-2022 17:10:25
basiccred_init-clust.tier1.kmeans.20.50	0.61	0.5733	0.5967	0.5933	0.6467	0.5933	13-May-2022 17:21:56
basiccred_init-clust.tier1.ica.100.Inf	0.6367	0.6167	0.5967	0.5933	0.6467	0.5933	13-May-2022 17:33:18
basiccred_init-clust.tier1.ica.100.50	0.63	0.6533	0.5967	0.5933	0.6467	0.5933	13-May-2022 17:49:27
basiccred_init-clust.tier1.ica.20.Inf	0.5833	0.5167	0.5967	0.5933	0.6467	0.5933	13-May-2022 17:58:04
basiccred_init-clust.tier1.ica.20.50	0.6767	0.6267	0.5967	0.5933	0.6467	0.5933	13-May-2022 18:12:37
basiccred_init-clust.tier1.icacrop.100.Inf	0.65	0.59	0.5967	0.5933	0.6467	0.5933	13-May-2022 18:24:34
basiccred_init-clust.tier1.icacrop.100.50	0.6633	0.5367	0.5967	0.5933	0.6467	0.5933	13-May-2022 18:41:19
basiccred_init-clust.tier1.icacrop.20.Inf	0.64	0.57	0.5967	0.5933	0.6467	0.5933	13-May-2022 18:50:44
basiccred_init-clust.tier1.icacrop.20.50	0.5967	0.6	0.5967	0.5933	0.6467	0.5933	13-May-2022 19:06:36
basiccred_init-clust.tier1.icacroplots.100.Inf	0.65	0.6167	0.5967	0.5933	0.6467	0.5933	13-May-2022 19:17:29
basiccred_init-clust.tier1.icacroplots.100.50	0.6367	0.6167	0.5967	0.5933	0.6467	0.5933	13-May-2022 19:32:48
basiccred_init-clust.tier1.icacroplots.20.Inf	0.6567	0.6067	0.5967	0.5933	0.6467	0.5933	13-May-2022 19:42:54
basiccred_init-clust.tier1.icacroplots.20.50	0.6733	0.6233	0.5967	0.5933	0.6467	0.5933	13-May-2022 19:57:19
metacredand_init-clust.tier1.kmeans.100.50-initmeta.10-clust.meta.kmeans.10.50	0.5567	0.55	0.5967	0.5933	0.6433	0.5933	13-May-2022 20:08:38
basiccredand_init	0.69	0.58	0.5967	0.5933	0.6433	0.5933	13-May-2022 20:15:41
basiccredand_init-clust.tier1.kmeans.100.Inf	0.6433	0.6	0.5967	0.5933	0.6433	0.5933	13-May-2022 20:22:44
basiccredand_init-clust.tier1.kmeans.100.50	0.6533	0.63	0.5967	0.5933	0.6433	0.5933	13-May-2022 20:29:56
basiccredand_init-clust.tier1.kmeans.20.Inf	0.64	0.64	0.5967	0.5933	0.6433	0.5933	13-May-2022 20:35:38
basiccredand_init-clust.tier1.kmeans.20.50	0.6267	0.5633	0.5967	0.5933	0.6433	0.5933	13-May-2022 20:44:12

analysis	tst_acc	trn_acc	pixel_tst_acc	pixel_trn_acc	edge_tst_acc	edge_trn_acc	timestamp
metacred_init-clust.tier1.kmeans.100.50-initmeta.10-clust.meta.kmeans.10.50	0.5767	0.5767	0.7367	0.7967	0.6867	0.7733	13-May-2022 14:58:22
metacred_init-clust.tier1.kmeans.20.50-initmeta.10-clust.meta.kmeans.10.50	NaN	NaN	0.7133	0.6767	0.7167	0.7933	13-May-2022 15:06:14
metacred_init-clust.tier1.ica.100.50-initmeta.10-clust.meta.kmeans.10.50	0.58	0.6367	0.71	0.78	0.6667	0.6833	13-May-2022 15:10:05
metacred_init-clust.tier1.ica.100.50-initmeta.10-clust.meta.ica.10.50	0.64	0.6067	0.6833	0.7633	0.6633	0.7667	13-May-2022 15:20:42
metacred_init-clust.tier1.icacrop.100.50-initmeta.10-clust.meta.kmeans.10.50	0.52	0.68	0.6967	0.81	0.72	0.77	13-May-2022 15:31:07
metacred_init-clust.tier1.icacrop.100.50-initmeta.10-clust.meta.icacrop.10.50	0.6033	0.52	0.7033	0.76	0.7	0.8333	13-May-2022 15:41:27
metacred_init-clust.tier1.icacroplots.100.50-initmeta.10-clust.meta.kmeans.10.50	0.63	0.7467	0.7067	0.8367	0.72	0.75	13-May-2022 15:52:11
metacred_init-clust.tier1.icacroplots.100.50-initmeta.10-clust.meta.icacroplots.10.50	0.67	0.6633	0.7	0.75	0.6667	0.7033	13-May-2022 16:03:20
metacred_init-clust.tier1.icacropsome.100.50-initmeta.10-clust.meta.kmeans.10.50	0.5533	0.51	0.6567	0.7667	0.6833	0.8333	13-May-2022 16:14:38
metacred_init-clust.tier1.icacropsome.100.50-initmeta.10-clust.meta.icacropsome.10.50	0.5733	0.61	0.7067	0.7433	0.6833	0.8233	13-May-2022 16:25:41
basiccred_init	0.7033	0.6933	0.6733	0.7867	0.6167	0.77	13-May-2022 16:36:49
basiccred_init-clust.tier1.kmeans.100.Inf	0.5	0.8167	0.69	0.7633	0.63	0.7333	13-May-2022 16:52:44
basiccred_init-clust.tier1.kmeans.100.50	0.6833	0.7733	0.7033	0.7367	0.6433	0.7167	13-May-2022 17:01:29
basiccred_init-clust.tier1.kmeans.20.Inf	0.6463	0.7283	0.684	0.7377	0.6973	0.7703	13-May-2022 17:10:25
basiccred_init-clust.tier1.kmeans.20.50	0.6153	0.677	0.7027	0.755	0.6913	0.806	13-May-2022 17:21:56
basiccred_init-clust.tier1.ica.100.Inf	0.7007	0.7553	0.6927	0.757	0.6933	0.7667	13-May-2022 17:33:18
basiccred_init-clust.tier1.ica.100.50	0.665	0.6377	0.708	0.7417	0.6913	0.7627	13-May-2022 17:49:27
basiccred_init-clust.tier1.ica.20.Inf	0.5723	0.6887	0.6917	0.7557	0.6887	0.7303	13-May-2022 17:58:04
basiccred_init-clust.tier1.ica.20.50	0.6523	0.5983	0.6853	0.773	0.693	0.8007	13-May-2022 18:12:37
basiccred_init-clust.tier1.icacrop.100.Inf	0.6593	0.7737	0.6877	0.7397	0.7123	0.7723	13-May-2022 18:24:34
basiccred_init-clust.tier1.icacrop.100.50	0.6263	0.6303	0.6993	0.739	0.688	0.7247	13-May-2022 18:41:19
basiccred_init-clust.tier1.icacrop.20.Inf	0.6203	0.6647	0.6663	0.7653	0.7003	0.8007	13-May-2022 18:50:44
basiccred_init-clust.tier1.icacrop.20.50	0.5763	0.5617	0.6703	0.7573	0.7053	0.77	13-May-2022 19:06:36
basiccred_init-clust.tier1.icacroplots.100.Inf	0.6973	0.7637	0.696	0.7207	0.674	0.72	13-May-2022 19:17:29
basiccred_init-clust.tier1.icacroplots.100.50	0.6557	0.7293	0.691	0.745	0.677	0.7667	13-May-2022 19:32:48
basiccred_init-clust.tier1.icacroplots.20.Inf	0.6747	0.701	0.699	0.764	0.693	0.7517	13-May-2022 19:42:54
basiccred_init-clust.tier1.icacroplots.20.50	0.6833	0.7037	0.683	0.756	0.678	0.79	13-May-2022 19:57:19
metacredand_init-clust.tier1.kmeans.100.50-initmeta.10-clust.meta.kmeans.10.50	0.5243	0.5927	0.69	0.7387	0.6793	0.735	13-May-2022 20:08:38
basiccredand_init	0.703	0.7037	0.6887	0.767	0.7047	0.7343	13-May-2022 20:15:41
basiccredand_init-clust.tier1.kmeans.100.Inf	0.6353	0.711	0.672	0.7437	0.67	0.7997	13-May-2022 20:22:44
basiccredand_init-clust.tier1.kmeans.100.50	0.703	0.692	0.673	0.7247	0.6817	0.796	13-May-2022 20:29:56
basiccredand_init-clust.tier1.kmeans.20.Inf	0.625	0.649	0.6797	0.7187	0.6483	0.6833	13-May-2022 20:35:38
basiccredand_init-clust.tier1.kmeans.20.50	0.6363	0.6307	0.696	0.718	0.6773	0.7327	13-May-2022 20:44:12

analysis	tst_acc	trn_acc	pixel_tst_acc	pixel_trn_acc	edge_tst_acc	edge_trn_acc	timestamp
basicimg_init-connected.25	0.4658	0.9956	0.6002	0.9978	0.5405	0.9978	13-May-2022 21:45:59

analysis	tst_acc	trn_acc	pixel_tst_acc	pixel_trn_acc	edge_tst_acc	edge_trn_acc	timestamp
basicimg_init-connected.25	0.549	0.9936	0.6461	0.9992	0.6295	0.9989	14-May-2022 00:14:48

11 References

1. Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078.

2. *Convolutional Neural Networks.* ibm.com. May, 2022 from https://www.ibm.com/cloud/learn/convolutio neural-networks.

3. *Deep Learning.* ibm.com. May, 2022 from https://www.ibm.com/cloud/learn/deep-learning.

4. Fukushima, K. (1980). Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. Biol. Cybernetics Vol. 36, pp. 193-202.

5. Graves, A., Wayne, G. & Danihelka, I. Neural Turing machines. http://arxiv.org/abs/1410.5401 (2014)

6. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

7. Han, C., Mao, J., Gan, C., Tenenbaum, J., & Wu, J. (2019). Visual concept-metaconcept learning. Advances in Neural Information Processing Systems, 32.

8. Johnson, J., Hariharan, B., Van Der Maaten, L., Fei-Fei, L., Lawrence Zitnick, C., & Girshick, R. (2017). Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2901-2910).

9. Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. Communications of the ACM, Vol. 60, no. 6, pp. 84-90.

10. LeCun, Y., Bengio, Y. & Hinton, G. E. (2015). Deep Learning. Nature, Vol. 521, pp 436-444.

11. LeCun, Y. et al. (1989) Backpropagation Applied to Handwritten Zip Code Recognition. Neural Computation, Vol. 1, no. 4, pp. 541-551.

12. Mao, J., Gan, C., Kohli, P., Tenenbaum, J. B., & Wu, J. (2019). The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. arXiv preprint arXiv:1904.12584.

13. Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. Advances in neural information processing systems, 26.

14. *Neural Networks.* ibm.com. May, 2022 from https://www.ibm.com/cloud/learn/neural-networks.

15. *Neuro-Symbolic AI.* MIT. May 2022 from https://mitibmwatsonailab.mit.edu/category/neuro-symbolic-ai/.

16. *Recurrent Neural Networks.* ibm.com. May, 2022 from https://www.ibm.com/cloud/learn/recurrent-neural-networks.

17. Sabour, Sara, Nicholas Frosst, and Geoffrey E. Hinton. "Dynamic routing between capsules." Advances in neural information processing systems 30 (2017).

18. Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

19. Shi, H., Zhang, Y., Chen, X., Tian, Y., & Zhao, J. (2019, September). Deep symbolic superoptimization without human knowledge. In International Conference on Learning Representations.

20. Sutskever, I., Martens, J., & Hinton, G. E. (2011, January). Generating text with recurrent neural networks. In ICML.

21. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).

22. Weston, J., Bordes, A., Chopra, S., Rush, A. M., Van Merriënboer, B., Joulin, A., & Mikolov, T. (2015). Towards ai-complete question answering: A set of prerequisite toy tasks. arXiv preprint arXiv:1502.05698.

23. Weston, M., Haudek, K. C., Prevost, L., Urban-Lurain, M., & Merrill, J. (2015). Examining the impact of question surface features on students' answers to constructed-response questions on photosynthesis. CBE—Life Sciences Education, 14(2), ar19.

24. *What is Deep Learning?* mathworks.com. May, 2022 from https://www.mathworks.com/discovery/deep-learning.html.

25. Yi, K., Wu, J., Gan, C., Torralba, A., Kohli, P., & Tenenbaum, J. (2018). Neural-symbolic vqa: Disentangling reasoning from vision and language understanding. Advances in neural information processing systems, 31.

26. Yi, K., Gan, C., Li, Y., Kohli, P., Wu, J., Torralba, A., & Tenenbaum, J. B. (2019). Clevrer: Collision events for video representation and reasoning. arXiv preprint arXiv:1910.01442.

12 Acknowledgements

This thesis would have been impossible without the help of Prof. Richard Granger, Eli Bowen, and Antonio Rodriguez. Prof. Granger graciously allowed me to participate in this project which he designed. Thank you as well to my advisor in the Computer Science department Soroush Vosoughi. I was humbled by my mentors' intelligence and commitment to a novel project.