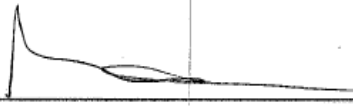


APPROVAL SHEET

Title of Thesis: Connecting Deep Neural Networks with Symbolic Knowledge

Name of Candidate: Arjun Kumar
Master of Science, 2016

Thesis and Abstract Approved: _____


Dr. Tim Oates
Professor
Department of Computer Science and
Electrical Engineering

Date Approved: _____

8/24/16

ABSTRACT

Title of thesis: **CONNECTING DEEP NEURAL NETWORKS
WITH SYMBOLIC KNOWLEDGE.**

Arjun Kumar, Master of Science, 2016

Thesis directed by: Dr. Tim Oates
Department of Computer Science and
Electrical Engineering

Neural networks have attracted significant interest in recent years due to their exceptional performance in various domains ranging from natural language processing to image identification and classification. Modern deep neural networks demonstrate state-of-the-art results in complex tasks such as epileptic seizure detection [14] and time series classification [18]. The internal architecture of these networks, in terms of learned representations, still remains opaque. This research addresses the first step in the long term motivation to construct a bi-directional connection between the raw input data and their symbolic representations. In this research, we examined whether a denoising autoencoder can internally find correlated principal features from input images and their symbolic representations which can be used to generate one from the other. Our results indicate that using symbolic representations along with the raw inputs generates better reconstructions. Our network was able to construct the symbolic representations from the input as well as input instances from their symbolic representations.

CONNECTING DEEP NEURAL NETWORKS WITH SYMBOLIC
KNOWLEDGE.

by

Arjun Kumar

Master's thesis submitted to the Faculty of the Graduate School of the
University of Maryland, Baltimore County in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
2016

Advisory Committee:
Dr. Tim Oates, Chair/Advisor
Dr. Charles Nicholas
Dr. Samuel Lomonaco

© Copyright by
Arjun Kumar
2016

Table of Contents

List of Figures	iii
1 Introduction	1
2 Background	4
2.1 Autoencoder	4
2.2 Deep Learning	8
2.3 Related Work	10
3 System Overview	13
3.1 Input Images	13
3.2 Autoencoder	14
3.3 Loss Function	15
3.4 Description Vector	17
3.5 Manipulations of the input	19
4 Experiments	21
4.1 Experiment 1: Basic Autoencoder test	23
4.2 Experiment 2: Description Vector Implementation	25
4.3 Experiment 3: Construction of Description Vector from Pixel Values .	27
4.4 Experiment 4: Construction of Image from Description Vector	31
4.5 Experiment 5: Setting the values for various Hyper-parameters	37
4.5.1 Batch Size	37
4.5.2 Learning Rate	38
5 Conclusion	40
Bibliography	42

List of Figures

2.1	Sample auto-encoder[1]	5
2.2	Sample denoising auto-encoder[16]	7
2.3	Sample Deep Neural Network[12]	10
3.1	Description Vector	18
3.2	Sample Input	19
3.3	Graph of function $y=\log(x)$	20
4.1	Training input instance and its reconstruction	23
4.2	Test input instance and its reconstruction	24
4.3	Training input instance and its reconstruction	25
4.4	Test input instance and its reconstruction	26
4.5	Test input instances	29
4.6	Variation of cross entropy values during testing phase for different percentage of pixels turned off	32
4.7	Standard deviation of cross entropy values during testing phase for different percentage of pixels turned off	33
4.8	Variation of Gaussian difference values during testing phase for different percentage of pixels turned off	33
4.9	Standard deviation of Gaussian difference values during testing phase for different percentage of pixels turned off	34
4.10	10% pixels turned off	35
4.11	20% pixels turned off	35
4.12	30% pixels turned off	35
4.13	40% pixels turned off	35
4.14	50% pixels turned off	36
4.15	60% pixels turned off	36
4.16	70% pixels turned off	36
4.17	80% pixels turned off	36
4.18	90% pixels turned off	37
4.19	100% pixels turned off	37
4.20	Graph (a) shows the variation of cross entropy values whereas graph (b) shows the variation in the mean pixel difference with change is the size of the batch.	38
4.21	Variation of cross entropy values per epoch during training phase for different learning rates	39

Chapter 1

Introduction

Deep neural networks are a form of artificial neural network which consists of many layers of hidden units between their input and output layers with the aim to capture a complex hierarchy of features of the inputs [10]. Using multiple hidden layers, deep neural networks extract features of inputs at multiple levels, allowing the network to learn complex mappings between these inputs and their expected outputs [2]. Recent advancements in the architecture and training mechanisms of deep neural networks have made them replace state-of-the-art systems in many fields.

Although deep neural networks present world class results in many domains, the internal representations these networks learn are still opaque. We intend to address this problem by trying to connect symbolic representations of the input to neural networks to understand and reason about what is learned by these networks.

An autoencoder is an artificial neural network which consists of identical input and output layers with one or more hidden layers which present themselves as limited capacity channels used to abstract complex features of the input space [4]. In its simplest form, an autoencoder first 'encodes' the input into a lower dimension latent representation and then tries to 'decode' these encodings back to their original form, learning the hierarchical mappings of the features of the inputs along the way [2].

There is prior work on studying the internal workings of deep neural networks by

visualizing pixels of the input images that cause activation of units in Deep Belief Networks [6] and feature maps in Convolutional Neural Networks [19]. CNNs have also been used in recent literature to generate images from their descriptions [5].

Systems have been developed to annotate images with a single sentence [11, 7] and even to describe the content of images with one or more sentences [17].

Using supervised training, deep neural networks have been used to refine horn clauses [15] and even derive new symbolic rules using classified examples [13].

This thesis is the first step into understanding how a deep neural network can relate to symbolic knowledge of the input. It contributes to the long term goal by investigating whether an autoencoder network can utilize the symbolic representations along with the raw input to produce better reconstructions. Further, it also determines whether the network can construct symbolic representations from the input images or input images from their symbolic representations.

In our work, we have trained a denoising autoencoder with blocks-world images and their symbolic representations to find the effect of these symbolic representations in terms of image reconstruction by the network. Each image in our input set consists of two blocks, one on top of the other. These images are represented symbolically in the form of a boolean vector which describe various aspects of the blocks and have bits on or off depending on the values for these aspects. Our results indicate that the autoencoder network when trained with input images and their symbolic representations is able to derive input images given their symbolic representations and also use input images to derive their symbolic representations.

Section 2 starts by describing the basics of an autoencoder and also introduces a

variation of the same used in this research. Further, this section gives an overview of deep learning and some insight into previous related work done. In section 3, we give a detailed description of our experimental setup describing the inputs used, the networks trained and various modifications done to the data. Section 4 presents results and analysis of various experiments and their results. Section 5 provides a summary of the work done in this thesis.

Chapter 2

Background

This chapter begins with an overview of an autoencoder before introducing the concept of a denoising autoencoder which we have used in our system. Further, the chapter includes a summary about deep learning and deep neural networks before concluding with a brief discussion on the previous work that relates to our objectives in this thesis.

2.1 Autoencoder

In this thesis, we trained an autoencoder to determine whether we can use symbolic representations along with the raw input to train the network, whether these symbolic representations can be reconstructed from the network, and to what degree the symbolic knowledge can be used to reconstruct the raw input itself. As introduced in [4], an autoencoder network's aim is reducing dimensionality via auto-association. As further explained in [4], the network achieves its goal by communicating the values for its input units to its output units, which are of the same size, through one or more hidden layers with a limited capacity bottleneck, encouraging the network to optimally encode the input vectors. These networks are trained to encode the input 'x' into some internal representation $f(x)$ in such a way that the input can be reconstructed from this representation. Hence, the target output for

an auto-encoder is its input [2].

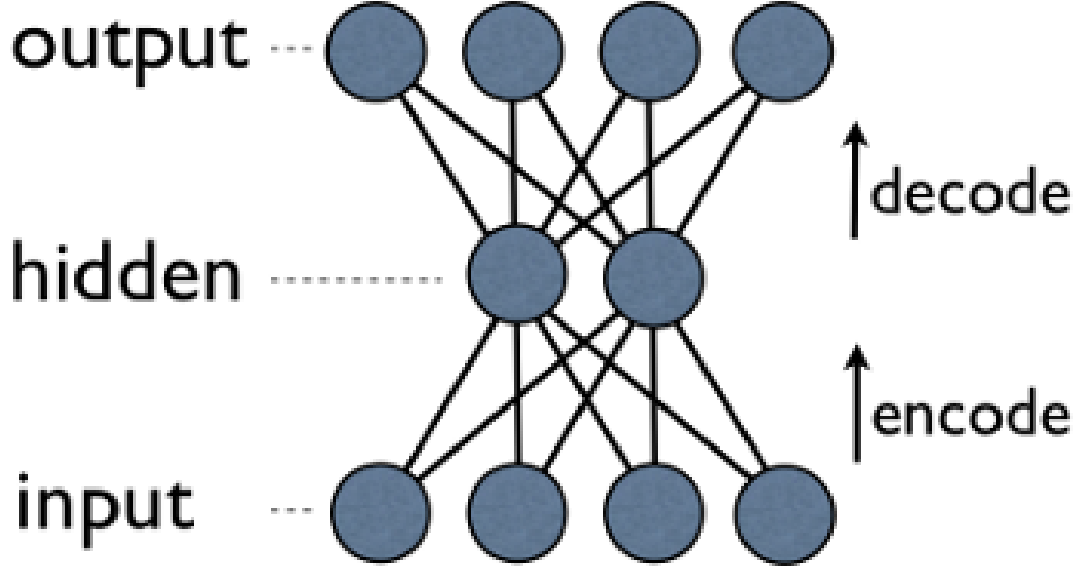


Figure 2.1: Sample auto-encoder[1]

As shown in figure 2.1, in its simplest form, an auto-encoder consists of two phases, an encoding phase followed by a decoding phase. In the first phase, the auto-encoder takes an input $\mathbf{x} \in [0, 1]^n$ and encodes it to a hidden representation $\mathbf{y} \in [0, 1]^{\acute{n}}$ by mapping it deterministically using $\mathbf{y} = f_{\theta}(\mathbf{x})$. A non-linear function, such as the sigmoid function $S(t) = \frac{1}{1+e^{-t}}$, is used as $f_{\theta}(x)$ parameterized by $\theta = [W, b]$ where 'W' represents the input weights of the network and is a $\acute{n} \times n$ matrix (where ' \acute{n} ' is the size of the hidden unit), whereas 'b' represents the input bias vector of size 'n'. Equation 2.1 summarizes the encoding phase (when a sigmoid function is used as the mapping function).

$$y = f_{\theta}(x) = \frac{1}{1 + e^{-(Wx+b)}} \quad (2.1)$$

The second phase of the auto-encoder, namely the decoding phase, takes the latent representation $\mathbf{y} \in [0, 1]^n$, passed as the output by the hidden layer(s), as its input and decodes it into the final output $\mathbf{z} \in [0, 1]^n$ which ideally is the reconstruction of the input given to the network initially in the first phase. This reconstruction is done via a deterministic mapping $\mathbf{z} = g_{\theta}(\mathbf{y})$ of the encoded representation of the input to its reconstruction using a non-linear function parameterized by $\theta = [\acute{W}, \acute{b}]$. Where ' \acute{W} ' and ' \acute{b} ' are the weight matrix and the bias vector of the decoding phase. The weight matrix of this second phase can be *tied* to the weight matrix of the previous phase by taking it as a transpose of the latter, i.e, $\acute{W} = W^T$. Equation 2.2 summarizes the decoding phase (when a sigmoid function is used as the mapping function).

$$z = g_{\theta}(y) = \frac{1}{1 + e^{-(\acute{W}y + \acute{b})}} \quad (2.2)$$

Various loss functions $L_H(x, z)$, as discussed in section 3.3, can be used to measure the reconstruction error between the input and the output of the network. Along with this loss function, stochastic gradient descent is used to train the network to produce better reconstructions. As mentioned in [2], the findings of [4] suggest that when the auto-encoder network has one hidden layer and uses the mean squared error criterion, the hidden layer of 'k' units learns to project the input on the first 'k' principle components of the input. Further, [2] also states that according to the findings in [8], making the hidden layer non-linear changes the behavior of an auto-encoder from PCA by giving the former the ability to capture multi-modal aspects of the input distribution.

The model discussed above is similar to the one used in [3] for building deep networks. The authors of [16] present a simple enhancement to this basic auto-encoder. The network is now trained with noisy input and is expected to reconstruct a clean output. Input \mathbf{x} is first corrupted, and then this corrupted input $\tilde{\mathbf{x}}$ is given to the network. The network is now trained to produce a noiseless output $\mathbf{z} = \mathbf{x}$. This is called a denoising auto-encoder. Equations 2.3 and 2.4 represent the encoding and decoding phase of a denoising auto-encoder network respectively ('s' represents a non-linear function like the sigmoid function).

$$y = f_{\theta}(\tilde{x}) = s(W\tilde{x} + b) \quad (2.3)$$

$$z = g_{\theta'}(y) = s(W'y + \acute{b}) \quad (2.4)$$

Image 2.2 gives a diagrammatic overview of a sample denoising auto-encoder network. The loss function evaluates the performance by comparing the reconstructed output with the clean input rather than the corrupted input that is fed into the network. q_D represents a function used to introduced noise into the original input.

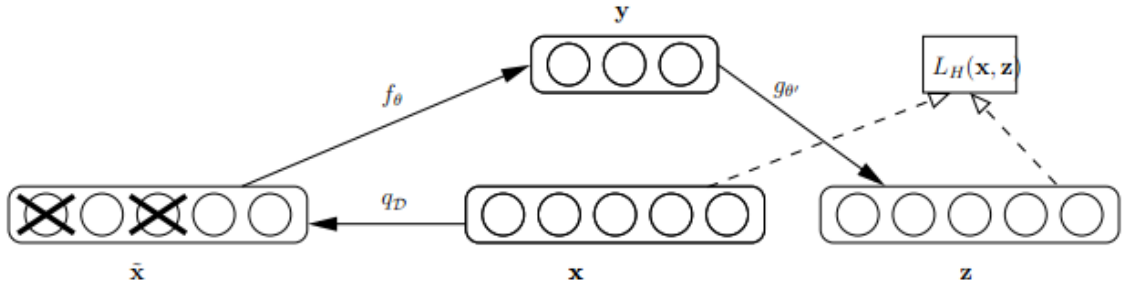


Figure 2.2: Sample denoising auto-encoder[16]

2.2 Deep Learning

As explained in [2], deep learning is a set of methods that typically use multi-layered artificial neural networks to learn multiple levels of representations. These representations correspond to a hierarchy of concepts where the higher levels of concepts are composed of lower level concepts in that hierarchy. Deep learning systems try to learn in multiple levels which corresponds to different levels of abstraction of the data [10]. Extracting features at multiple levels allows deep learning systems to learn complex mappings between the input and the expected output with little dependence on features that humans come up with for the system. [2]. As further explained in [10], the work done in deep learning methods can be classified into the following three categories based on the type of task the method is intended to accomplish:

- Deep Networks for unsupervised learning:

When used for unsupervised learning, the purpose of these networks is to analyze or synthesize patterns in the input by capturing high-order correlation of features in the data.

- Deep Networks for supervised learning:

When used for supervised learning, these networks are intended to break-down the hierarchy of concepts in the input in order to provide high discriminating power for classifying the data into various target classes which are provided directly or indirectly to the network.

- Hybrid Deep networks:

These types of networks can come in two flavors. First, where an unsupervised deep learning method is used to assist the classification task of the overall supervised categorization model. Second, where the classification capabilities of the supervised deep learning methods is used to estimate the parameters in the overall generative model.

The perceptron is one of the most basic types of machine learning agents and is inspired to mimic a neuron firing when the input received from the activation units reaches a threshold. To mimic the activities of a human brain, a collection of neurons are chained together to form a network consisting of one input layer and one output layer along with a small amount of hidden layers, usually 1 between them. All these layers are formed by one or more perceptrons tied together in a complex feed-forward system. A deep neural network consists of multiple hidden layers between the input and the output layer. This multilayer network has a set of fully connected weights which are sometimes initialized by supervised or unsupervised pre-training techniques. Figure 2.3 depicts a deep neural network with 3 hidden layers between an input and the corresponding output layer.

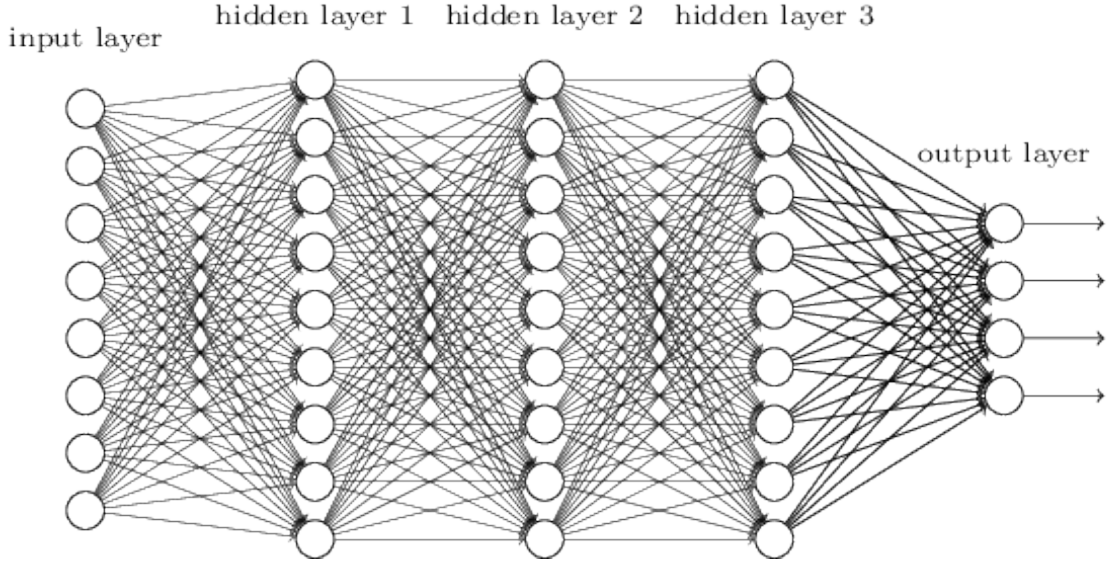


Figure 2.3: Sample Deep Neural Network[12]

2.3 Related Work

There is prior work on visualization of a deep network. Although studying visualization of the first-layer representations is very common in the literature, recent works have started to go beyond the first layer in understanding a neural network. The authors of [6] present an analysis of Stacked Denoising Autoencoders and Deep Belief Networks by visualizing the pixels at each layer of the network that maximize the activation of a given unit. A similar approach was taken in [19] for CNNs where the authors used a reverse convolutional network, Deconvnet, as mentioned in [19], to map the feature activities of the intermediate layers of the CNN back to the input state with the goal of finding input patterns that originally caused a given feature map activation. We intend to investigate the internal workings of a deep neural network by trying to find how and to what degree the final weights and bias of a

trained deep neural network relate to symbolic knowledge about the input space provided to it at the beginning of the training phase along with the raw instances. This thesis is the first step towards that goal.

Research in the field of computer vision worked on generating descriptions from an image. The first system to annotate images was presented in [11] where a collection of simple images and their descriptions was given to the network at training. The complex images were then divided into regions. These simple regions accumulated words from previously seen images into a vector which was quantized to produce the final result. A much later system presented in [7] parses an object, an action and a scene from an image and uses this 'triple' to retrieve a sentence from a pool of sentences written for similar images. Among the most recent works in this area is the research presented in [17] that uses a deep CNN and a recurrent neural network to annotate images with one or more sentences. Research presented in [9] uses a deep auto-encoder network to map images to short binary codes based on the levels of features extracted by the network. These binary codes are then combined with semantic hashing to find similar images. We have used description vectors, as described in section 3.4, to give symbolic details of the input images with each instance to the network. One of the experiments performed in this work is to test whether the auto-encoder network is able to construct these description vectors for the input images.

Work has been done regarding refining and deriving horn clauses using neural networks. Most noteworthy among them is the system called KBANN [15] which works by inserting a few hand written symbolic rules into a neural network, then refines

these rules by training it in a supervised fashion with classified examples resulting in a highly-accurate classifier network. The authors have extended their work in [13] to add the capability to their network, KBANN, to dynamically add new horn clauses. Our description vectors are represented with propositional logic. Our network is expected to learn meaningful representations of the input images which are able to reconstruct these description vectors along with the images.

The authors of [5] have presented a deep deconvolutional architecture which takes type, camera position and additional transformation values as tuples and produces an image of a chair. We explore the concept of generating images from their symbolic representations in this thesis by testing the percentage of input image our network is able to reconstruct using just the description vector for that instance.

Chapter 3

System Overview

This chapter gives a detailed description of our experimental setup before we talk about the actual experiments and their results in section 4 below. We used a denoising autoencoder to reconstruct colored blocks-world images and their symbolic representations. We begin this chapter by describing the images that are given as input to the network in section 3.1. Section 3.2 presents the specifications of the autoencoder used while section 3.3 gives an overview of a few loss functions considered. Along with the input images, we passed to the network a boolean vector which describes the physical attributes of the content of these input instances. Section 3.4 gives a detailed overview of how this symbolic representation is created for each input instance. Finally, section 3.5 concludes this chapter by presenting a few manipulations done to the input data before it is passed to the autoencoder.

3.1 Input Images

The input consists of simple blocks-world images of size 28 x 32 pixels. Each image consists of two blocks, one paced on top of the other. The blocks differ from each other based on the values of the following aspects:

1. Shape

Each block can take any shape from:

(a) Cone

(b) Box

(c) Sphere

2. Size

Relatively, the size of each block can be:

(a) Small

(b) Medium

(c) Large

3. Color

The blocks are colored singularly with one of the following colors¹:

(a) Red

(b) Green

(c) Blue

3.2 Autoencoder

The network consists of a denoising autoencoder with the following aspects:

1. Input Layer - Each input instance given to the network consists of pixel values of a single image. Therefore, the input layer has a total size of

$$3 \quad \times \quad (28 \times 32) \quad = \quad 2,688$$

¹Two blocks of same color have the same shade of that color in all instances.

(Values of the Red, Green	(Size of each
and Blue component)	image in pixels)

2. Hidden Layer - The network has one hidden layer of size 2000.

We conducted experiments, using the cross entropy measure (equation 4.2) to evaluate the results, to determine the size of the hidden layer of the network. Our experiments showed that instances of our network with hidden layer of size less than 1950 resulted in underfitting, whereas instances with hidden layer of size greater than 2800 resulted in overfitting. Therefore, we selected from among the lowest set of values from the range that produced the optimum results regarding these sets of experiments as the size of the hidden layer for our network.

3. Output Layer - Since the network tries to regenerate the input, the size of the output layer is same as that of the input layer.

3.3 Loss Function

A loss function quantifies how good or bad the network is performing in terms of the target output at the point where this function is encountered, in any phase of learning. For example, the loss function can quantify the network's performance after processing of every n instance(s) during the training phase and after processing of every m instance(s) during the testing phase. Based on the measure returned by this loss function, the parameter(s) of the network are updated during the training phase to drive it towards making more generalized predictions over the instance

space. Our network encounters the loss function after processing every mini-batch of instances during the training phase and after processing every instance during the testing phase. The network's performance is assessed based on the input image instance and its reconstruction produced by the network. While setting up the network, we considered the following loss functions for evaluating the performance of our network:

1. Cross Entropy: This measure is a combination of the entropy of input (i.e, \mathbf{x}) given the received output (i.e, \mathbf{z}) summed with the relative entropy of the input with respect to the output. When used as a loss function, cross entropy aims to increase the likelihood of receiving the desired output for each given input.

$$C_{CE}(z, x) = - \sum_k (x_k \ln(z_k) + (1 - x_k) \ln(1 - z_k)) \quad (3.1)$$

2. Sum of Squared Errors: This measure sums up the squares of deviations of the actual data received from the expected data. In case of an autoecoder, we expect the output of the network to be the input itself. Therefore, when used as a loss function, this measure aims to increase the similarity of the output received to the inputs given to the network.

$$C_{SSE}(z, x) = \sum_k (z_k - x_k)^2 \quad (3.2)$$

3. Exponential : This measure can emulate the simplicity of the sum of squared error measure and also the non-linearity of the cross entropy measure. When

used as a loss function, this measure adds more robustness against outliers to the system.

$$C_{EXP}(z, x) = \tau \exp\left(\frac{1}{\tau} \sum_k (z_k - x_k)^2\right) \quad (3.3)$$

In equations 3.1, 3.2 and 3.3 above, 'k' represents the total number of instances in a particular set over which the measure is used. \mathbf{x}_k and \mathbf{z}_k represent the k^{th} input instance and its respective reconstruction produced by the autoencoder.

We chose to use the cross entropy measure to be used as a loss function as our experiments indicated that it outperformed the other two in terms of Gaussian difference between pixel values (explained in section 4) of the input and its respective reconstruction.

3.4 Description Vector

The symbolic knowledge is provided to the network in the form of a vector. This vector is concatenated with each input instance and provides information to the network regarding the content of that instance. Figure 3.1 describes this input vector.

Bit	Property	Description		Value
0	Medium	Size of the block.	Block on top	0 / 1
1	Big			0 / 1
2	Small			0 / 1
3	Red	Color of the block.		0 / 1
4	Green			0 / 1
5	Blue			0 / 1
6	Cone	Shape of the block.		0 / 1
7	Box			0 / 1
8	Sphere			0 / 1
9	Medium	Size of the block.	Block on bottom	0 / 1
10	Big			0 / 1
11	Small			0 / 1
12	Red	Color of the block.		0 / 1
13	Green			0 / 1
14	Blue			0 / 1
15	Cone	Shape of the block.		0 / 1
16	Box			0 / 1
17	Sphere			0 / 1

Figure 3.1: Description Vector

As shown in figure 3.1, the description vector is eighteen bits long. It uses the first nine bits to describe the block on the top and the last nine bits to describe the block on the bottom. Each of these nine bits sets consists of three units of three bits each describing the size, color and the shape of the block respectively. Each of these sets of three bits is called an aspect set. Each aspect set has only one bit on while the other two are off. The bit that is on corresponds to the value that the object possesses for that particular aspect. For example, if the top object is of size *big*, only bit 1 will be on, while bits 0 and 2 will be off. The following example illustrates how the complete description vector is created for a sample image

Figure 3.2 consists of a Medium Blue Sphere on top of a Big Green Box. To describe the block on the top, bits 0, 5 and 8 are set indicating that this block has the

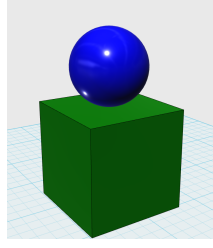


Figure 3.2: Sample Input

values Medium, Blue and Sphere for the size, color and shape aspects respectively.

Hence, the description vector half for this block will be 100 001 001.

Similarly, for the block at the bottom, bits 10, 13 and 16 are set indicating that it is a Big Green Box 010 010 010.

Therefore, the complete description vector for this input figure is

100 001 001 010 010 010.

3.5 Manipulations of the input

The red, green and blue content values of each pixel was divided by 255 to bring each value of the image vector between 0-1. This was done so that these values could be used with the log function in the Cross Entropy Error function.

The description vector of the input images contains 0 in most places. This yields an infinite number exception when used with the log function. To mitigate this problem, each value in the description vector was increased by 0.1.

For example, the description vector for figure 3.2 in section 3.4 (100 001 001 010 010 010) will now become

1.1 0.1 0.1 0.1 0.1 1.1 0.1 0.1 1.1 0.1 1.1 0.1 0.1 1.1 0.1 0.1 1.1 0.1

As shown in figure 3.3 below, the log function shows maximum difference (is most sensitive) in the y-axis for values 0.25 and 0.75 on the x-axis. Therefore, the input vectors and the description vectors were multiplied by 0.25 and 0.75 respectively in addition to the manipulations mentioned previously. Continuing with our example, finally the description vector that is sent as input to the network is

$$(1.1 \ 0.1 \ 0.1 \quad 0.1 \ 0.1 \ 1.1 \quad 0.1 \ 0.1 \ 1.1 \quad 0.1 \ 1.1 \ 0.1 \quad 0.1 \ 1.1 \ 0.1 \quad 0.1 \ 1.1 \ 0.1) \times$$

$$0.75 = 0.825 \ 0.075 \ 0.075 \quad 0.075 \ 0.075 \ 0.825 \quad 0.075 \ 0.075 \ 0.825 \quad 0.075 \ 0.825 \ 0.075$$

$$0.075 \ 0.825 \ 0.075 \quad 0.075 \ 0.825 \ 0.075$$

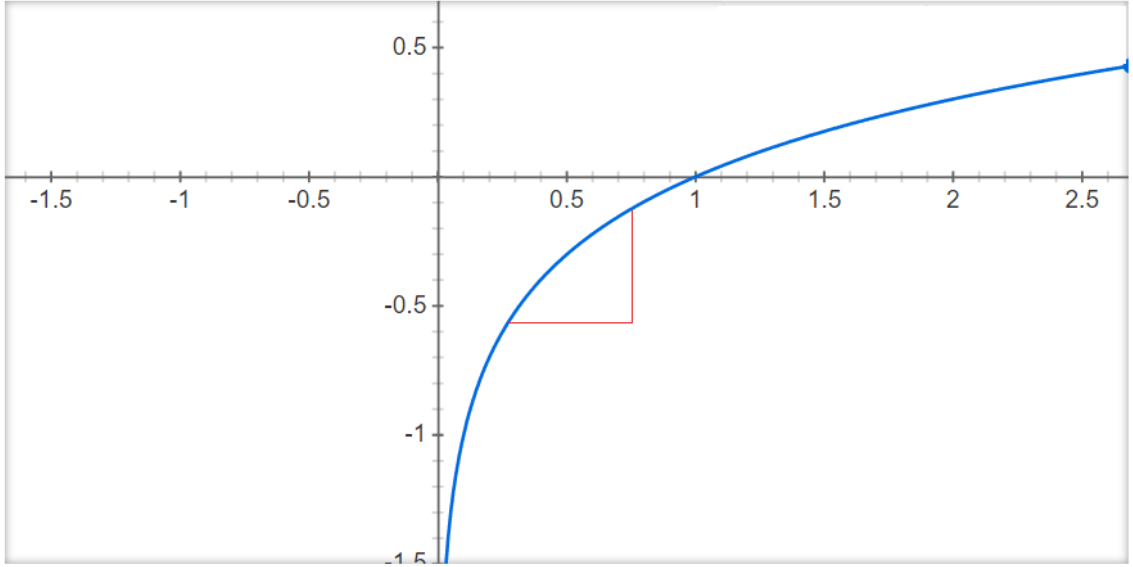


Figure 3.3: Graph of function $y=\log(x)$

Chapter 4

Experiments

This chapter describes the experiments that we performed, the evaluation metrics used to analyze the results, and the conclusions drawn from them. The inputs to the network are taken as described in section 3.1 above. Out of this input set, 15 percent of instances are randomly chosen to be included in the testing set, whereas the other 85 percent form the training set. A description vector is then created for each instance as described in section 3.4. Thereafter, the training and testing set instances are randomized to change their positions in their respective set. Finally, the manipulations mentioned in section 3.5 are applied to each instance before starting the training and testing phase.

Result evaluation metrics:

- Gaussian difference between pixel values

For each input instance, its reconstructed output is represented in the network in the form of a vector of size 2688. We then take a Gaussian difference between these two vectors to measure the similarity between the input instance and its reconstruction produced by the network. A mean of this Gaussian difference is calculated across the training and testing sets along with standard deviation of the instances for each set. Equation 4.1 below is a mathematical representation of the Gaussian difference between an input instance and its corresponding

output.

$$C_{GD}(z, x) = \frac{1}{n} \sum_n \sqrt{(\|Pn_z - Pn_x\|)^2} \quad (4.1)$$

'n' refers to the size of the instance vector. Pn_x refers to the n^{th} bit in the input instance whereas Pn_z represents the n^{th} bit in the reconstruction vector produced by the network for that instance.

- Cross Entropy error between input and output instances

To measure how well a model has performed, average cross entropy error is calculated between input instances and their respective reconstructions generated by the network for both the training and testing sets. As mentioned in section 3.3, Cross Entropy is also used as a loss function to train the network. When used as a result evaluation metric, the cross entropy loss is calculated for all the instances in the final epoch during the training phase and then the mean of this loss value is calculated over the whole training set. When using this metric to calculate the performance for the testing set, the input instances are encoded and then decoded back using the trained network weights and biases. The cross entropy is then calculated between the input test instance and its representation constructed by the network which again is averaged over the whole testing set. This metric can be summarized by equation 4.2 below

$$C_{CE}(z, x) = - \sum_k (x_k \ln(z_k) + (1 - x_k) \ln(1 - z_k)) \quad (4.2)$$

In the equation above, 'x' and 'z' represent an input instance and its reconstruction produced by the network respectively. Since this measure is calculated over a complete set (training set or testing set) at one time, 'k' refers

to a particular instance in that set. Hence x_k and z_k represent the k^{th} input instance and its reconstruction in a set. A mean of this cross entropy error is calculated across the training and testing set along with standard deviation of the instances for each set.

These evaluation metrics are used to analyze the results of experiments 4.1, 4.2, 4.4 and 4.5. Evaluation metrics for 4.3 are described in section 4.3 itself.

4.1 Experiment 1: Basic Autoencoder test

Aim: The aim of this experiment is to run a simple denoising autoencoder to find a baseline to evaluate further experiments.

Manipulations: The input consisted of only the image pixel values and no description vector was added.

Result: Image 4.1 below shows a sample training instance and its reconstruction produced by the autoencoder.

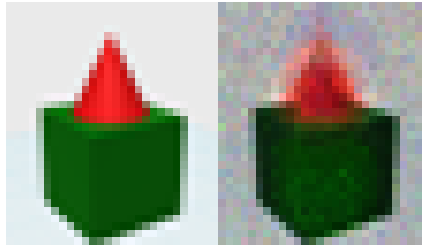


Figure 4.1: Training input instance and its reconstruction

The values for Cross Entropy and Gaussian Difference obtained over the complete training set are as follows:

Cross Entropy

Mean Value: 1199.174497

Standard Deviation: 72.71138798

Gaussian Difference

Mean Value: 1.101731193

Standard Deviation: 0.2576063209

Image 4.2 below shows a sample test instance and its reconstruction produced by the autoencoder.

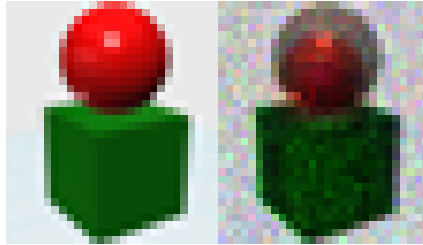


Figure 4.2: Test input instance and its reconstruction

The values for Cross Entropy and Gaussian Difference obtained over the complete test set are as follows:

Cross Entropy

Mean Value: 1149.818023

Standard Deviation: 105.1876571

Gaussian Difference

Mean Value: 1.081603678

Standard Deviation: 0.3212884685

This experiment was done to get results from the network by using only the raw

input and no symbolic representations of the input were added. We found that the mean values for both the cross entropy and Gaussian difference were similar over the testing and the training set. This suggests that the network is stable. Comparing the standard deviation values for both the metrics between the train and test sets suggests that although the network is able to reconstruct these sets equally as a whole, it is not able to reconstruct some instances as good as the others in the test set.

4.2 Experiment 2: Description Vector Implementation

Aim: The aim of this experiment is to test whether the autoencoder is able to reconstruct the input images when the description vector for each instance is concatenated to it.

Manipulations: A description vector, as explained in section 3.4, is created and concatenated to each instance before giving it as an input to the network.

Result: Image 4.3 below shows a sample training instance and its reconstruction produced by the autoencoder.

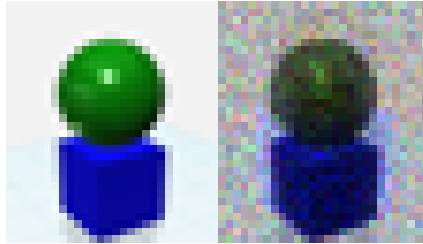


Figure 4.3: Training input instance and its reconstruction

The values for Cross Entropy and Gaussian Difference obtained over the com-

plete training set are as follows:

Cross Entropy

Mean Value: 1202.462661

Standard Deviation: 71.73612782

Gaussian Difference

Mean Value: 1.105054967

Standard Deviation: 0.1757282312

Image 4.4 below shows a sample test instance and its reconstruction produced by the autoencoder.

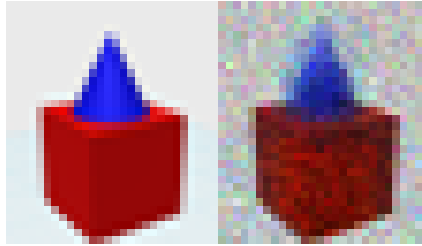


Figure 4.4: Test input instance and its reconstruction

The values for Cross Entropy and Gaussian Difference obtained over the complete test set are as follows:

Cross Entropy

Mean Value: 1178.745283

Standard Deviation: 121.5242428

Gaussian Difference

Mean Value: 1.095618494

Standard Deviation: 0.163169406

Comparing the results mentioned above with those of experiment 4.1 we saw that the value for Cross Entropy increases when the description vectors are also included along with the image pixel values. This is attributed to the network now having 18 more values per instance to reconstruct. The fact that the Gaussian Difference between an image and its reconstruction is almost similar in both the experiments suggests that including description vectors in the input does not hamper the performance of the network. Rather, looking at the standard deviation of the Gaussian Difference values between the two networks, it can be said that with the help of the description vectors, the network can now make better predictions for even those instances for which it was not able to do so previously in experiment 4.1.

4.3 Experiment 3: Construction of Description Vector from Pixel Values

Aim: The aim of this experiment is to test whether the network is able to construct the description vector from the pixel values of the input image.

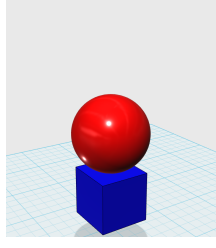
Manipulations: The training input instances consist of values for both the image vector and the description vector. The test input instances however, consist of only the image vector, with all the values turned off in the description vector, i.e., the last 18 values of the input consists of all zeros.

Result evaluation metrics:

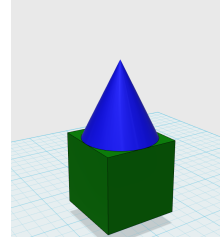
Maximum Pooling Hamming Distance

As mentioned in section 3.4, each set of three bits is used to describe one aspect of the block and only one bit among these three is set indicating the value the block holds from the three possible values for that aspect. The output consists of the description vector constructed by the network using the pixel values from the input image only. Each aspect set of three bits in the output however might not consist of two zeros bits and one set bit. Maximum pooling is done to find the highest value bit in each aspect set of this description vector. These bits are then compared with the maximum pooled bits for each aspect set from the corresponding expected description vector for each instance and a Hamming distance is calculated between the two description vectors to evaluate how many aspect sets are constructed accurately by the network. For example, if the size of the bottom block is *Big*, then the values $[0.1 \ 0.6 \ 0.2]$ or $[0 \ 0.4 \ 0]$ for bits nine, ten and eleven respectively are considered correct whereas value $[0.6 \ 0.2 \ 0.1]$ is considered incorrect for these bits in the output description vector. The Hamming distance of maximum pooled bits of the description vectors is calculated for each instance in the set and then a mean is calculated across the entire set along with the standard deviation of the instances that belong to the set.

Examples:



(a) Input instance 1



(b) Input instance 2

Figure 4.5: Test input instances

Input figure 4.5a consists of a medium red sphere on a small blue box. The expected description vector for this instance is

$$(0.075 \ 0.825 \ 0.075 \quad 0.825 \ 0.075 \ 0.075 \quad 0.075 \ 0.075 \ 0.825 \quad 0.075 \ 0.075 \ 0.825 \\ 0.075 \ 0.075 \ 0.825 \quad 0.075 \ 0.825 \ 0.075)$$

The description vector constructed by the network for this instance is

$$(0.3 \ 0.4 \ 0. \quad 0.7 \ 0. \ 0. \quad 0.1 \ 0. \ 0.4 \quad 0. \ 0. \ 0.6 \quad 0. \ 0. \ 0.7 \quad 0. \ 0.7 \ 0.)$$

Since each aspect set has the maximum value in the correct bit position, this description vector is considered to be constructed correctly.

Whereas, the instance in figure 4.5b is a medium blue cone on a medium green box.

The expected description vector for this instance is

$$(0.825 \ 0.075 \ 0.075 \quad 0.075 \ 0.075 \ 0.825 \quad 0.825 \ 0.075 \ 0.075 \quad 0.825 \ 0.075 \ 0.075 \\ 0.075 \ 0.825 \ 0.075 \quad 0.075 \ 0.825 \ 0.075)$$

The description vector constructed by the network for this instance is

$$(0.2 \ 0. \ 0.3 \quad 0. \ 0. \ 0.5 \quad 0.4 \ 0. \ 0.1 \quad 0.7 \ 0. \ 0. \quad 0. \ 0.4 \ 0. \quad 0. \ 0.7 \ 0.)$$

Among the first three bits, bit 0 should have the highest value indicating that the size of the top block in the instance is of size medium. Instead, bit number two has

the highest value among the first three in the constructed vector. Therefore, for this instance, the description vector generated by the network is considered to have a Hamming distance of 2 with the expected description vector.

Result: In our experiments, we had sixteen test instances. Therefore, the network was expected to construct $16 \times 6 = 96$ aspect sets correctly. In the results we got a total error of 8 aspect sets which were constructed incorrectly over the whole test set. Out of these incorrect aspect sets, 62.5% were for the size, 25% were for the color and 12.5% were for the shape of the object. Thus, each description vector constructed by our network had an average Hamming distance of $(8 \times 2) / 16 = 1$ from the expected description vector. Therefore we can infer from the results of this experiment that the network is able to symbolic descriptions for the images from their pixel values with an accuracy of $88 / 96 = 91.67\%$. On further evaluation of the results we found that when the network made an error in predicting the value for an aspect set depicting the size of the object, it was uncertain of the values it generated for the three bits in that set. This is evident in the example mentioned above where the network generates values 0.2 0. and 0.3 for the aspect set depicting the size of the block on the top. In contrast to this, when the network made an error in predicting the value for an aspect set depicting the color or the shape of the object, it was certain of the incorrect values it generated for the three bit is the set. For example, in one of the test instances, the network was expected to generate an aspect set (0.075 0.075 0.825) for the shape of the block depicting it to be a sphere. The network incorrectly generated the value (0.5 0.0 0.1) for that respective aspect set being certain of the incorrect value.

4.4 Experiment 4: Construction of Image from Description Vector

Aim: The aim of this experiment is to test whether the network is able to construct the image from the description vector of the input instance.

Manipulations: Starting from 10 percent, pixels of the input images are turned off in the increments of 10 percent for each iteration of the experiment. The description vector of each instance is provided to the network.

To make the network more sensitive to description vectors, equation 4.2 mentioned above was modified to equation 4.3 hence penalizing the network more for when the reconstruction of the description vector compared with the related image pixel values.

$$C_{CE}(z, x) = - \sum_k (\alpha \times (x_i \ln(z_i) + (1 - x_i) \ln(1 - z_i)) + (x_d \ln(z_d) + (1 - x_d) \ln(1 - z_d))) \quad (4.3)$$

In the equation above, 'x' and 'z' represent an input instance and its reconstruction produced by the network respectively. Since this measure is calculated over a complete set (training set or testing set) at one time, 'k' refers to a particular instance in that set. Each instance is represented as a vector consisting of by 'i' bits for representing the pixel values and 'd' bits representing the values of the description vector for that instance. A mean of this cross entropy error is calculated across the training and testing set along with standard deviation of the instances for each set.

Result: Graphs 4.6 and 4.7 show the mean and standard deviation values respectively for the cross entropy value between testing instances and their related outputs received from the network in variation with the percentage of pixels turned off in

each repetition of experiment. Graphs 4.8 and 4.9 similarly represent the Gaussian difference readings received.

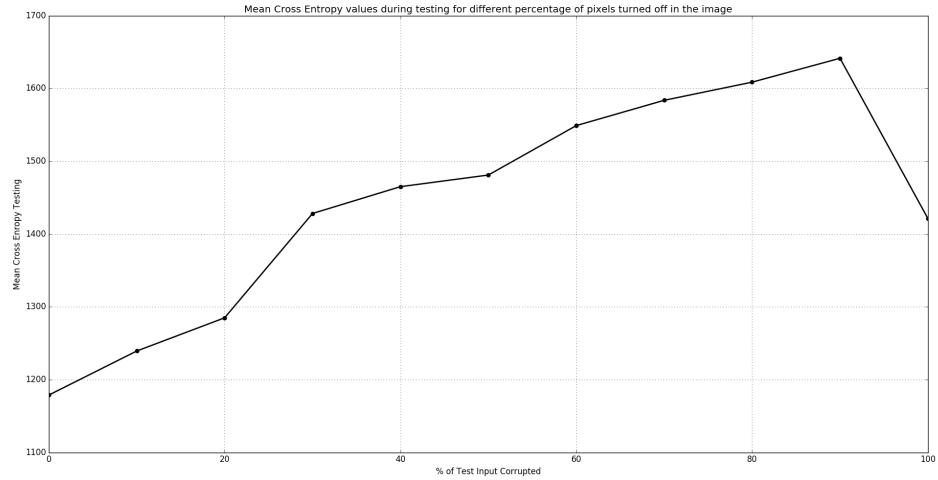


Figure 4.6: Variation of cross entropy values during testing phase for different percentage of pixels turned off

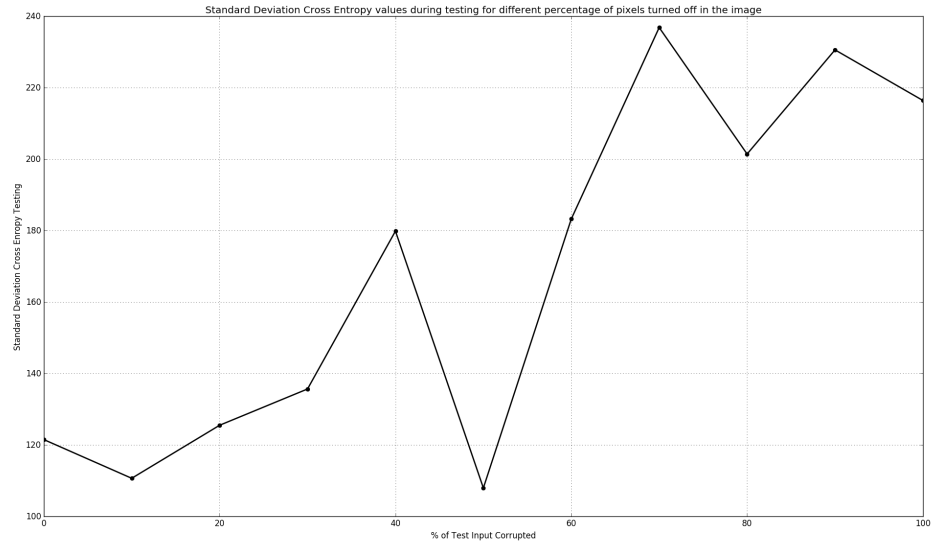


Figure 4.7: Standard deviation of cross entropy values during testing phase for different percentage of pixels turned off

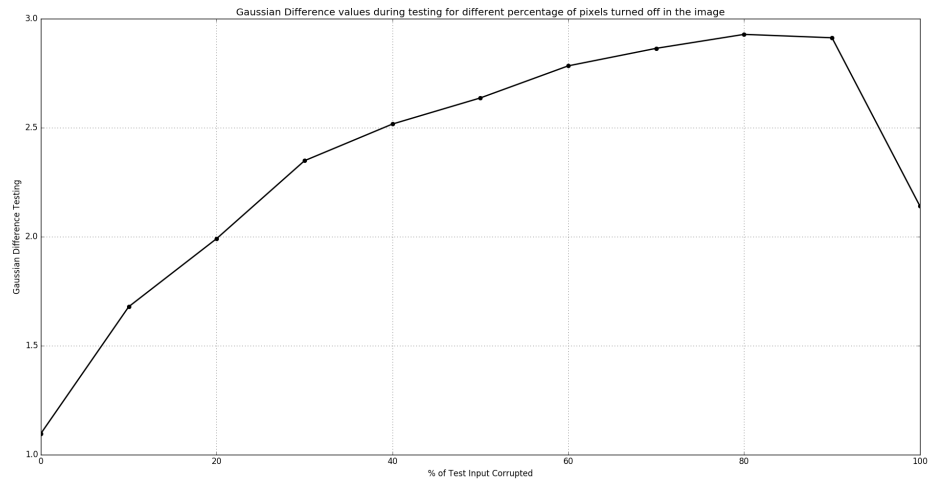


Figure 4.8: Variation of Gaussian difference values during testing phase for different percentage of pixels turned off

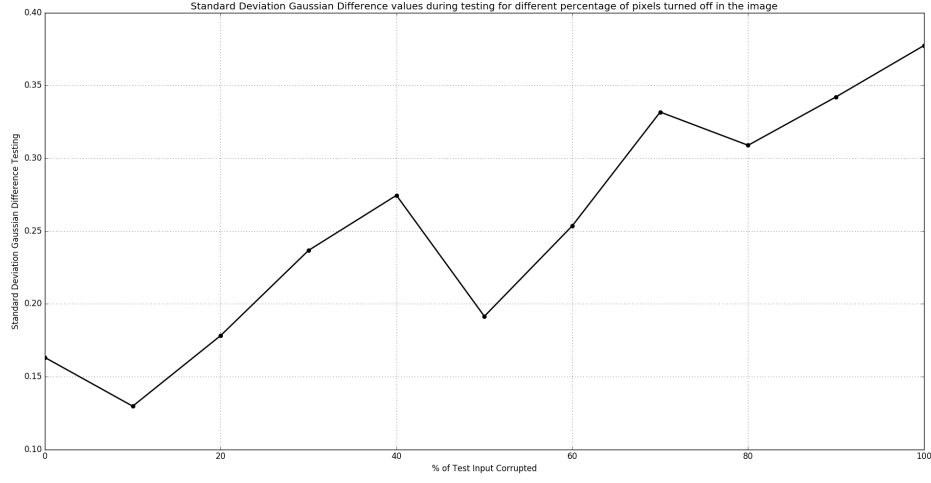


Figure 4.9: Standard deviation of Gaussian difference values during testing phase for different percentage of pixels turned off

The graphs above suggest that the network is able to reconstruct the images when only some (0%-30%) of the pixels are turned off. Image reconstructions become more lossy when the percentage of pixels that are turned off goes up from 40% to 90%. When all the pixels are turned off, the network is able to produce reconstructions that are less lossy than those produced by turning only 30% of the pixels off. We attribute this behavior of the network to the modification done to basic cross entropy method (i.e., equation 4.3) which enables the network to focus more on learning the description vectors and find connections to generate image pixels using the same. Therefore we can infer from the results of this experiment that the network is able to construct images from their symbolic descriptions.

Figure 4.10 to 4.19 below show an original test instance concatenated with the actual test instance given to the network with a certain percentage of pixels turned

off followed by its reconstruction generated by the network.



Figure 4.10: 10% pixels turned off

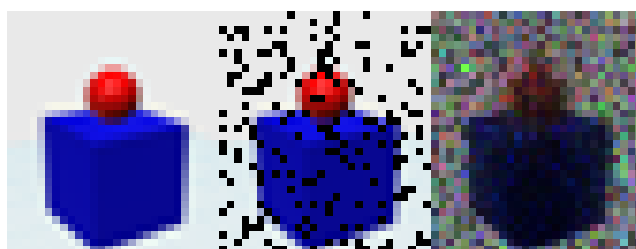


Figure 4.11: 20% pixels turned off



Figure 4.12: 30% pixels turned off



Figure 4.13: 40% pixels turned off

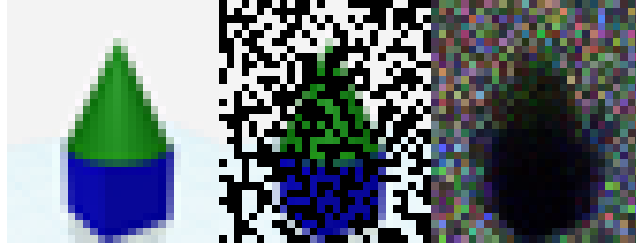


Figure 4.14: 50% pixels turned off



Figure 4.15: 60% pixels turned off



Figure 4.16: 70% pixels turned off



Figure 4.17: 80% pixels turned off



Figure 4.18: 90% pixels turned off

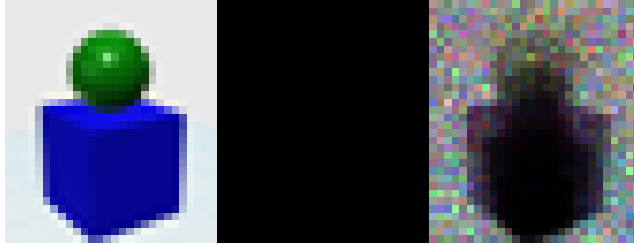


Figure 4.19: 100% pixels turned off

4.5 Experiment 5: Setting the values for various Hyper-parameters

The following experiments were done to determine the different hyper-parameters which would yield the best input reconstruction from the network. The results demonstrate the effect of varying the values of these parameters and the selection criteria for their optimal values.

4.5.1 Batch Size

Aim: The aim of this test is to determine the size of the batch that the autoencoder should process before updating its weight and bias matrices.

Result:

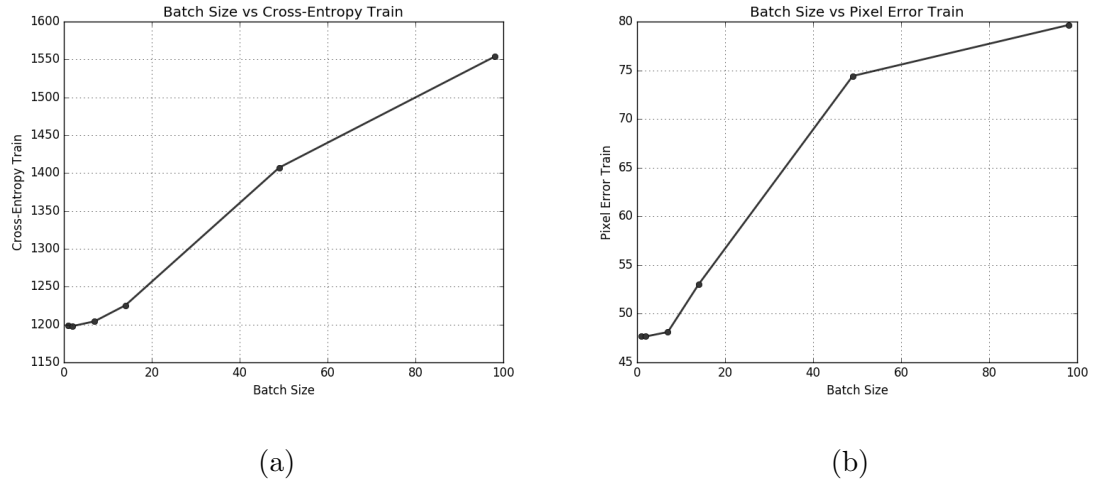


Figure 4.20: Graph (a) shows the variation of cross entropy values whereas graph (b) shows the variation in the mean pixel difference with change is the size of the batch.

In our experiments, as seen in graph 4.20a and 4.20b, batch of size '2' gives the optimum values for both cross entropy as well as pixel difference between the input and output images. Hence we used a batches of size '2' in our experiments.

4.5.2 Learning Rate

Aim: The aim of this test is to determine the rate at which the network should update its weight matrices and bias vectors.

Result:

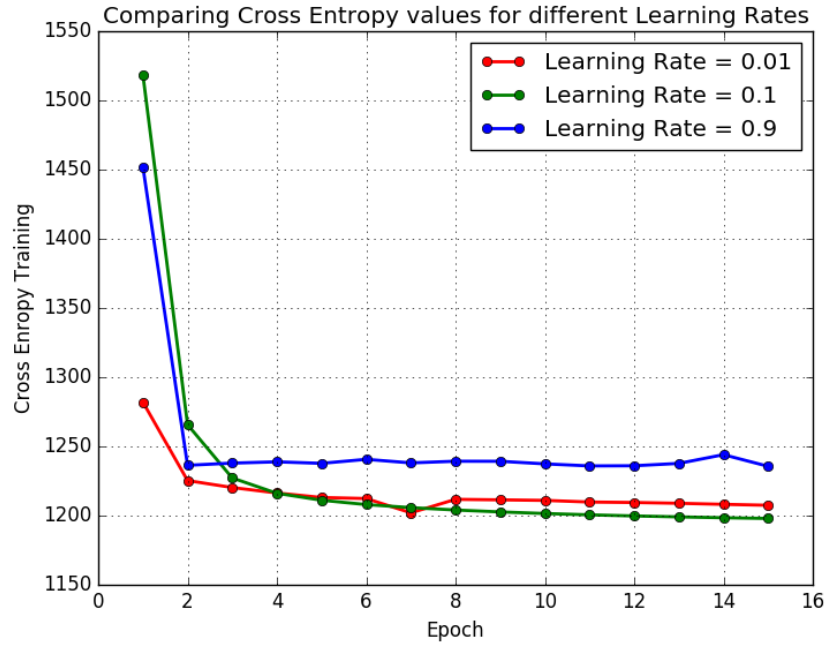


Figure 4.21: Variation of cross entropy values per epoch during training phase for different learning rates

As seen in graph 4.21 above, keeping the learning rate 0.1 reduces the cross entropy as the number of epochs increase in the training phase. Therefore we kept the learning rate as '0.1' in our experiments.

Chapter 5

Conclusion

With their capability to learn multiple levels of representations corresponding to a hierarchy of concepts of the input domain, deep neural networks have replaced state of the art systems in multiple domains. The next step in understanding these multi-layered complex systems is to analyze their internal workings. In this thesis, we presented the first step in the long term commitment to construct a bi-directional connection between the raw input data and their symbolic representations.

An autoencoder tries to optimally encode its inputs into their reduced dimension latent representations and then tries to successfully decode these representations to reconstruct the provided input.

We trained an autoencoder by using images along their symbolic representations to find whether the network can internally represent images and their symbolic representations in the form of their principal features that can not only reconstruct the image and also whether it can correlate the principle features of the image to that of its symbolic knowledge to reconstruct one from another. Our inputs consisted of blocks-world images and we represented the symbolic knowledge of these input images in the form of a description vectors. The results of our experiments indicate the following:

- The network when trained with input images and their description vectors

was able to construct description vectors for test instances from only the pixel values of the images.

- Similarly, the network when trained with input images and their description vectors was able to construct image pixel values for test instances from only their description vectors.
- Description vectors passed to the autoencoder along with the input images enabled it to find less lossy latent representations for more images in the input space.

Bibliography

- [1] Cross validated: "what is the difference between convolutional neural networks, restricted boltzmann machines, and auto-encoders?".
- [2] Yoshua Bengio. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.
- [3] Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, et al. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153, 2007.
- [4] Hervé Bourlard and Yves Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59(4-5):291–294, 1988.
- [5] Alexey Dosovitskiy, Jost Tobias Springenberg, and Thomas Brox. Learning to generate chairs with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1538–1546, 2015.
- [6] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341, 2009.
- [7] Ali Farhadi, Mohsen Hejrati, Mohammad Amin Sadeghi, Peter Young, Cyrus Rashtchian, Julia Hockenmaier, and David Forsyth. Every picture tells a story: Generating sentences from images. In *European Conference on Computer Vision*, pages 15–29. Springer, 2010.
- [8] Nathalie Japkowicz, Stephen Jose Hanson, and Mark A Gluck. Nonlinear autoassociation is not equivalent to pca. *Neural computation*, 12(3):531–545, 2000.
- [9] Alex Krizhevsky and Geoffrey E Hinton. Using very deep autoencoders for content-based image retrieval. In *ESANN*, 2011.
- [10] Dong Yu Li Deng. Deep learning: Methods and applications. Technical report, 2014.
- [11] Yasuhide Mori, Hironobu Takahashi, and Ryuichi Oka. Image-to-word transformation based on dividing and vector quantizing images with words. In *First International Workshop on Multimedia Intelligent Storage and Retrieval Management*, pages 1–9. Citeseer, 1999.
- [12] Michael A. Nielsen. *Michael A. Nielsen, "Neural Networks and Deep Learning"*. Determination Press, 2015.

- [13] David W Pitz and Jude W Shavlik. Dynamically adding symbolically meaningful nodes to knowledge-based neural networks. *Knowledge-based systems*, 8(6):301–311, 1995.
- [14] Siddharth Pramod, Adam Page, Tinoosh Mohsenin, and Tim Oates. Detecting epileptic seizures from eeg data using neural networks. *arXiv preprint arXiv:1412.6502*, 2014.
- [15] Geoffrey G Towell and Jude W Shavlik. Knowledge-based artificial neural networks. *Artificial intelligence*, 70(1):119–165, 1994.
- [16] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.
- [17] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3156–3164, 2015.
- [18] Zhiguang Wang and Tim Oates. Encoding time series as images for visual inspection and classification using tiled convolutional neural networks. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.
- [19] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833. Springer, 2014.

