Application of Approximate Message Passing in Neural Networks

A Project Report

submitted by

PARIKSHIT SHESHACHALA HEGDE

in partial fulfilment of the requirements

for the award of the degree of

BACHELOR OF TECHNOLOGY AND MASTER OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

MAY 2019

THESIS CERTIFICATE

This is to certify that the thesis titled **Application of Approximate Message Passing in Neural Networks**, submitted by **Parikshit Sheshachala Hegde**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology and Master of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Andrew Thangaraj
Research Guide
Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: IIT Madras, Chennai

Date: 5th May 2019

ACKNOWLEDGEMENTS

I would like to start by thanking my thesis advisor Dr. Andrew Thangaraj. Throughout the whole project, he has devoted a large amount of his valuable time in helping me with my project. His passion and drive for doing cutting-edge research inspires me, and I hope to inculcate his passion, dedication and discipline. The last one to two years has been the biggest learning experience in my life, and a huge chunk of it is thanks to Dr. Thangaraj.

I would also like to thank Dr. Rahul Vaze from TIFR, with whom I did two summer internships. Those were one of my first research projects, and Dr. Vaze was extremely patient with me as I slowly picked up the skills and developed the mindset to do research. It was under him that I first realized that I enjoy research. I would also like to thank Dr. Kaushik Mitra and Dr. B Ravindran, with whom I did a couple of extremely interesting projects in machine learning. I would also like to thank each and every professor who taught me. The classes were extremely engaging, and they ensured that I always had a desire to learn more.

I owe deeply to all the friends that I have made in the last five years. From gaming all night in the first couple of years to having academic discussions all night in the last couple of years, we have matured together. I will forever remember the fond memories that we have created. I owe a special gratitude to my friend Harikumar. We have done most of our courses together, and I couldn't have asked for a better study partner. We also worked together for our final Master's theses.

Finally, I would like to thank my parents and my brother. In the last 5 years, they gave me the freedom to make my own decisions, and then encouraged and supported each one of my decisions.

ABSTRACT

KEYWORDS: Approximate Message Passing; Neural Networks.

The problem of performing inference using generative Neural-Network(NN) models is considered. Particularly, the problem of recovering an image given a partial information about it is considered, where the partial information could be because of factors such as noise added to the image, occlusion in the image, etc. We use Approximate Message Passing(AMP) algorithms, which are designed to perform inference on linear models, on the NN's to perform the task. Although NN's represent non-linear functions, we show that AMP can be used to perform inference on them. We illustrate using examples that the performance of AMP algorithms is considerably good on tasks such as occlusion removal, scattered inpainting and noise removal.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS i

ABSTRACT ii

LIST OF FIGURES v

ABBREVIATIONS vi

NOTATION vii

1 INTRODUCTION 1

2 Background: Neural Networks 4

2.1 Machine Learning 4

2.2 Neural Networks 5

2.3 Auto-Encoder(AE) 6

2.3.1 Experiment on MNIST 7

2.4 Variational Auto-Encoder(VAE) 8

2.4.1 Experiment on MNIST 10

3 Background: Approximate Message Passing 11

3.1 AMP for Compressed Sensing 11

3.2 Generalized Approximate Message Passing(GAMP) 13

4 AMP on Synthetic Neural Network 16

4.1 Synthetic Neural Network Construction 16

4.2 Inference on Synthetic Neural Network 17

4.2.1 Recover $\mathbf{h_1}$ from $\mathbf{Y}$ 17

4.2.2 Recover $\mathbf{X}$ from $\hat{\mathbf{h}}_\mathbf{1}$ 18

4.2.3 Experiment 18

5 AMP on Real Neural Network 21

5.0.1 Neural Network Construction 21
5.1 Inference using GAMP 21
5.1.1 Adjustment to the Weight Matrices 23
5.1.2 Inference Experiment 24
5.2 Applications 25
5.2.1 Application 1: Occlusion Removal 25
5.2.2 Application 2: Scattered Inpainting 26
5.2.3 Application 3: Noise Removal 28

6 Conclusion and Future Work **30**

LIST OF FIGURES

1.1 Motivation Example 1

2.1 A 4-6-6-6-4 Neural Network 6
2.2 ReLU and Sigmoind Activation Functions 6
2.3 An example AE 7
2.4 AE on MNIST Results. 8
2.5 AE Latent Space Encoding 9
2.6 An Example VAE 9
2.7 VAE on MNIST Results. 10
2.8 VAE Latent Space Encoding 10

3.1 GAMP Observation Model 14
3.2 GAMP Algorithm 14

4.1 Synthetic Network Model 16
4.2 An example run of GAMP on Synthetic Network 19
4.3 AE on MNIST Results. 20

5.1 VAE Architecture 21
5.2 Histogram of Weight Matrices of the Decoder 23
5.3 Histogram of modified weight matrix $W_d^{(2)\prime}$. 24
5.4 Results from estimation using GAMP 24
5.5 Occlusion Removal task model. 25
5.6 Image Estimation given Occluded Image Observation. 26
5.7 Scattered Inpainting Task model. 27
5.8 Dynamics of GAMP on Scattered Inpainting Task. 27
5.9 Dynamics of GAMP on Occlusion Task. 28
5.10 Image Estimation given scattered missing points. 28
5.11 Image Estimation given Noisy Observation. 29

ABBREVIATIONS

AE	Auto Encoder
AMP	Approximate Message Passing
GAMP	Generalized Approximate Message Passing
KL-Divergence	Kulback-Leibler Divergence
MAP	Maximum-a-Posteriori
MMSE	Minimum Mean Squared Error
NN	Neural Network
ReLU	Rectified Linear Unit
VAE	Variational Auto Encoder

NOTATION

$\mathbf{z}$	Latent Vector of AE/VAE
$\mathbf{x}$ or $\mathbf{X}$	Input to the neural network
$\mathbf{y}$ or $\mathbf{Y}$	Output to the Neural Network
$\hat{\mathbf{x}}$ or $\hat{\mathbf{X}}$	Output of AE/VAE
W	Weight matrix of a neural network
f	Function represented by neural network
f_e	Function represented by encoder part of AE/VAE
f_d	Function represented by decoder part of AE/VAE
$p()$	Probability Distribution
μ	Mean of a random variable
σ^2	Variance of a random variable
$\mathcal{N}(\mu, \sigma^2)$	Normal distribution with mean μ and variance σ^2
Unif$[a, b]$	Uniform distribution in the range a to b
$\mathbb{E}[\cdot]$	Expectation of a random variable
$\min$	Minimum value of a function

Unless otherwise mentioned, bold-faced symbols indicate vectors.

Capital-lettered symbols will be used to represent random variables and matrices.

CHAPTER 1

INTRODUCTION

Neural Networks have demonstrated remarkable success in wide-ranging applications such as image-recognition, speech-recognition, object-detection,autonomous-driving, speech-translation and many more [1–5]. One promising line of work in this area is that of the use of neural networks as generative models for data. Let's consider the example of a neural network that is a generative model for real-world images. It takes a random-vector $\mathbf{z}$(usually i.i.d., gaussian vector is used in literature) as an input, and then outputs a vector $\mathbf{x}$ that looks like a real-world image. Usually, the dimension of $\mathbf{z}$ is much smaller than that of $\mathbf{x}$. If we were to consider $\mathbf{z}$ to be i.i.d., uniform, information theorists can recognize this to be very similar to the problem of source coding or data compression, where the goal is to use a uniform random vector of the smallest size possible to represent a probabilistic source(in this example, the source is the set of real-world images). However, the techniques used by the neural-networks community to accomplish this task of finding a generative model is very different from that used to perform source coding by the information theory community. We will discuss these techniques briefly in the chapter 2.

The question that we ask now is: Can the generative model be used to perform inference on the data it has been trained on. Consider the example in the Figure 1.1. On the left is the original image of the number 7. On the right is the same image, with the

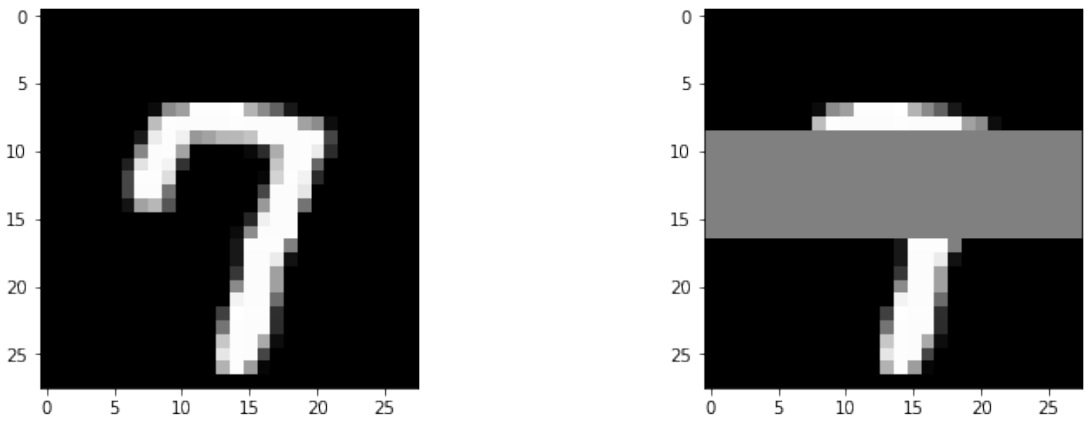


Figure 1.1: Motivation Example

middle-section being occluded. Our goal is to now retrieve the original image, given that we only observe the occluded image. In order to think about this problem, let's write it in a mathematical form. Let the observation model be $\mathbf{y} = A\mathbf{x}$, where $\mathbf{x}$ is N-length vector representing the original image (vectorized), the matrix A represents the observation model, and $\mathbf{y}$ is the observed vector of length n. In the case of the occlusion example above, the matrix A can be constructed by first considering the $N \times N$ identity matrix, and then deleting the rows corresponding to the pixels that have been occluded. The problem is to now try to recover $\mathbf{x}$ according to the following optimization problem:

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}} L(\mathbf{x}, \mathbf{y}), \tag{1.1}$$

$L(\cdot, \cdot)$ in the above equation is a loss function, which could take one of several forms. Considering $L(\mathbf{x}, \mathbf{y}) = -p(\mathbf{x}|\mathbf{y})$ leads to the widely used MAP estimate. Another popular choice for the loss function is, $L(\mathbf{x}, \mathbf{y}) = \int ||\mathbf{x} - \tilde{\mathbf{x}}||^2 p(\tilde{\mathbf{x}}|\mathbf{y}) d\tilde{\mathbf{x}}$, which leads to the MMSE estimate. $p(\mathbf{x}|\mathbf{y})$ denotes the posterior distribution of $\mathbf{x}$ given the observation $\mathbf{y}$. Using Bayes rule, it is often written as $p(\mathbf{x}|\mathbf{y}) \propto p(\mathbf{x})p(\mathbf{y}|\mathbf{x})$, where $p(\mathbf{x})$ is the prior distribution of $\mathbf{x}$, and $p(\mathbf{y}|\mathbf{x})$ describes the observation model. As a side note, setting $p(x) \propto e^{-\lambda|x|}$ and making the observation model an AWGN: $p(y|x) \sim Ax + \mathcal{N}(0, \sigma^2)$, makes this problem equivalent to the compressed sensing problem [6, 7]. We will talk about this in a little bit of detail in chapter 3.

Let's now come back to the problem of recovering the whole image from the occluded image. This problem poses some immediate challenges: firstly, since the space of images is so complex, it is very difficult to succinctly describe the prior $p(x)$(on the other hand, $p(y|x)$ can be easily described since occlusion is a simple process). Several techniques have been invented in the computer science literature to overcome this challenge by finding several approximations to this prior. One of the techniques that has achieved a lot of success in recent years is to use a generative neural network to model the prior. Techniques have been developed to perform MAP and MMSE estimation using a generative model as a prior [8].

One major downside to the above techniques is that they are computationally very intensive. To address this issue, we turn to an algorithm called Approximate Message Passing(AMP) which has been used previously in the context of compress sensing and

signal estimation under linear observation models. This algorithm is explained briefly in chapter 3. Since a neural network is a highly non-linear model, AMP cannot be applied directly. But, all hope is not lost, since a neural network is composed of linear layers followed by non-linear operations. We will discuss the techniques that we use in more detail in chapter 4 and chapter 5. AMP has rigorous theoretical guarantees when the transformation matrix has i.i.d., gaussian entries. We thus first illustrate our techniques on a synthetic neural network with i.i.d., gaussian entries in chapter 4. We then show in chapter 5, that AMP works on real neural networks as well(which do not have i.i.d., gaussian weight matrices), by illustrating its effectiveness on 3 applications: occlusion, scattered inpainting and noise removal.

CHAPTER 2

Background: Neural Networks

Deep Learning(or Neural Networks) refers to a subset of algorithms used in Machine Learning. So, we will first start by briefly introducing Machine Learning.

2.1 Machine Learning

Several applications in real-life require computers to make decisions given certain inputs. For example, given an image, can the computer decide whether it is an image of a cat or a dog? More generally, given an input x, can we find a function $f(x)$ keeping in mind a certain loss function(in the previous example, $f(x)$ should correctly classify pictures of cats and dogs). Unlike many problems in computer science, here it is very difficult for a human to design an algorithm that follows a set of logical rules, because dogs and cats come in so many shapes, sizes and colours! Machine Learning addresses this problem by taking a data-driven approach. Let's describe this in some mathematical detail.

Data is being generated as $X, Y \sim p(x, y)$, where $X \in \mathcal{X}$ is a d-dimensional random variable representing the input, and $Y \in \mathcal{Y}$ is a 1-dimensional random variable representing the output. Now, given m samples: $\{X_i, Y_i\}_{i=1}^m$, all generated i.i.d., according to $p(x, y)$, the goal is to learn a function $f : \mathcal{X} \to \mathcal{Y}$, such that the expected loss is minimized:

$$\min_f \mathbb{E}\left[L(f(X), Y\right], \tag{2.1}$$

where $L(\cdot, \cdot)$ is an appropriate loss function. If Y is a continuous random variable, which is the case in the problems that we will consider, it is common to choose $L(f(X), Y) = (f(X) - Y)^2$. Let's write the machine learning algorithm as $\mathcal{A} : (\mathcal{X}, \mathcal{Y})^m \to \mathcal{F}$, where $\mathcal{F}$ denotes the set of functions that the algorithm can learn. $\mathcal{F}$ could take several forms. One of the most basic forms that is considered is the set of

all linear mappings $F = W^T X$, where W is a matrix of appropriate dimensions. Under this class of functions, notice that (2.1) is a convex optimization problem, and thus the global optimum can easily be computed. This algorithm is called as linear regression, in literature. Clearly, linear functions are not powerful enough to represent complex mappings from X to Y(such as in the example of cats and dogs). Neural Networks are basically another set of functions $\mathcal{F}$ and we will discuss them next.

2.2 Neural Networks

We will only consider a simple feed-forward fully connected neural network model. It is composed of neurons, which perform the following computation: $\mathbf{x} \mapsto \sigma(\mathbf{w}^T\mathbf{x} + b)$, where $\mathbf{w}, b$ are parameters and $\sigma(\cdot)$ is a scalar activation function. Two popular activation functions used in literature are sigmoid() and ReLU() which are described in Figure 2.2. These neurons are arranged in parallel layers, so that the output of each layer can be compactly represented as $\mathbf{x} \mapsto \sigma(W^T\mathbf{x} + \mathbf{b})$, where W is a matrix(each column corresponding to the parameter vector of one of the neurons), $\mathbf{b}$ is a vector, and σ is applied point-wise to the vector $W^T\mathbf{x} + \mathbf{b}$. These layers are now stacked on top of each other, thus producing a function of the form:

$$f(\mathbf{x}) = \sigma_k(W_k^T\sigma_{k-1}(W_{k-1}^T \dots \sigma_2(W_2^T\sigma_1(W_1^T\mathbf{x} + \mathbf{b_1}) + \mathbf{b_2}) + \dots \mathbf{b_{k-1}}) + \mathbf{b_k}), \quad (2.2)$$

where $W_i, \mathbf{b_i}, \sigma_i$ are parameters of the i-th layer of the neural network. The number of layers k is denoted as the depth of the neural network, and the number of columns in W_i is called the width of the layer i. A visual interpretation of a neural network is shown in Figure 2.1.

With this complicated function, the optimization problem (2.1) is no longer convex. However, gradient-based methods have been shown to find local-minima that still show a good performance. The simplest of these methods is gradient descent, which has the following updates: $\mathbf{w_{t+1}} = \mathbf{w_t} - \eta_t \nabla \mathbb{E}\left[L(f(X), Y)\right]$, where η_t is the learning rate. Recently, more sophisticated gradient-based algorithms have been developed such as

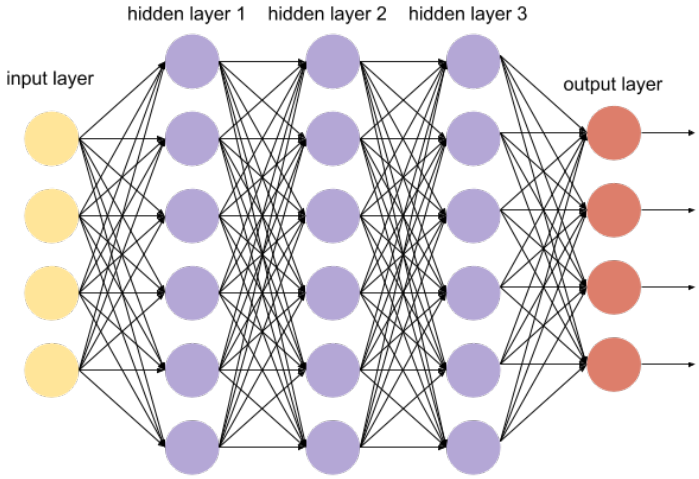


Figure 2.1: A 4-6-6-6-4 Neural Network [1]

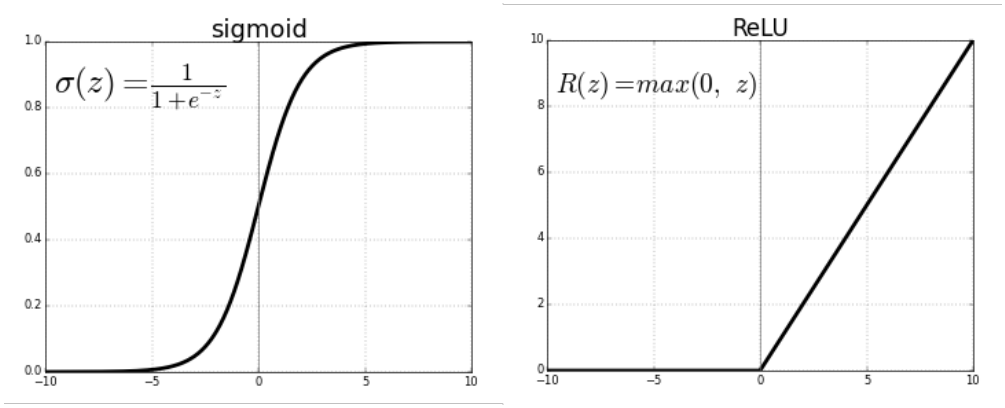


Figure 2.2: ReLU and Sigmoid Activation Functions [2]

AdaGrad and Adam, which have better convergence properties [9, 10].

As highlighted in the Introduction, our goal is to construct a generative model using a neural network. So, we will now briefly discuss two such popular models.

2.3 Auto-Encoder(AE)

In order to train a generative model, we will first train a model that learns to map $\mathbf{x} \mapsto \mathbf{x}$. The neural network can be thought of as being composed of two networks: an encoder f_e and a decoder f_d. A visualization of this network has been shown in Figure 2.3. Therefore, the encoder first transforms $\mathbf{x}$ to a compact representation, $\mathbf{z} = f_e(\mathbf{x})$. We

[2]Image Courtesy: https://github.com/drewnoff/spark-notebook-ml-labs/tree/master/labs/DLFramework

[2]Image Courtesy: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

define $d_z = \dim(z)$, and usually $d_z << d$. Then, the decoder tries to recover the vector $\mathbf{x}$ by doing $\hat{\mathbf{x}} = f_d(z)$. The space occupied $\mathbf{z}$ is called the latent space. The whole network is thus represented as:

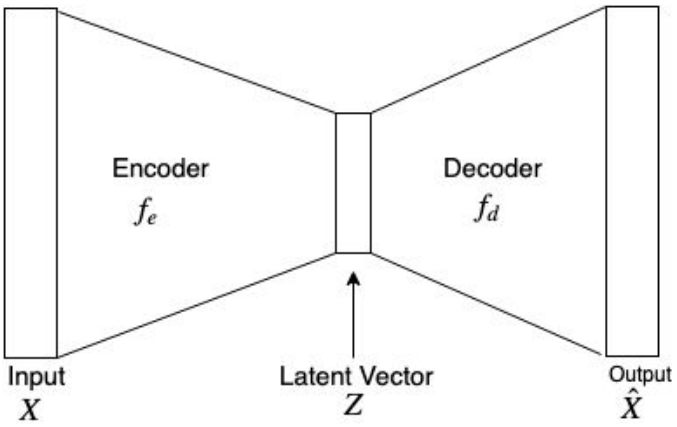


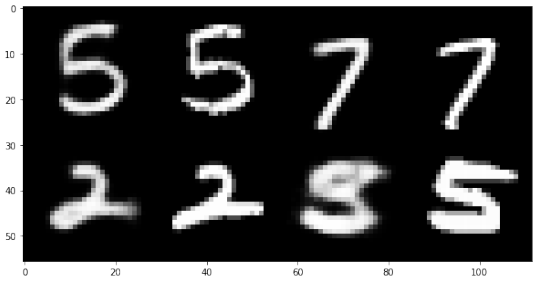
Figure 2.3: An example AE

$$\hat{\mathbf{x}} = f_d(f_e(x)).$$

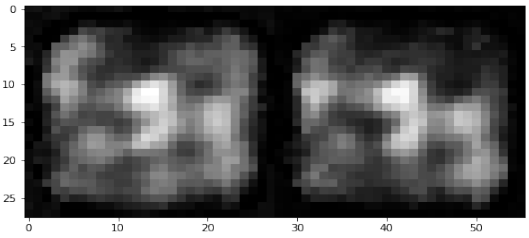
For the training, the whole network(encoder and decoder) is trained together. This model is called a standard Auto-Encoder(AE) [11]. Now, to use an AE as a generative model, one can simply discard the encoder network, and only work with the decoder network f_d. On being input a random vector $\mathbf{z}$ to it, the decoder $f_d(\mathbf{z})$ should hopefully produce a meaningful output that is likely to be produced by the data-generation process $p(X)$. Let us now perform a small experiment on the popular MNIST data-set and analyze the shortcomings of the standard AE model.

2.3.1 Experiment on MNIST

The MNIST dataset consists of 60,000 gray-scale images of digits, where each image is of size 28×28 [12]. The images are vectorized to vectors of size 784×1 before inputting it to the network. For the first part of the experiment, the network architecture that was chosen was $784 - 256 - 128 - 64 - 16$ for the encoder and $16 - 64 - 128 - 256 - 784$ for the decoder(the numbers refer to the width of the corresponding layer of the neural network). Therefore, the AE tries to summarize each input of size 784 by a vector of size 20. The model was trained using the Adam Optimizer with default parameters



(a) Mapping between input(on the left) and the output produced by AE(on the right).



(b) Images generated by AE from random latent vector

Figure 2.4: AE on MNIST Results.

on TensorFlow [13]. Figure 2.4 illustrates the results obtained. The figure on the left shows how an image $\mathbf{x}$(in the first and third column) gets mapped to $\hat{\mathbf{x}}$(in the 2nd and 4th column). This illustrates that the AE has been able to learn the map $\mathbf{x} \mapsto \hat{\mathbf{x}}$ quite well. The figure in the right of 2.4, shows the images $\hat{\mathbf{x}}$ produced by the decoder of the AE, upon being input an i.i.d., gaussian random latent vector $\mathbf{z}$. It is clear that these images do not look like digits at all. So, AE's cannot be directly used as generative models for our purposes.

To understand the failure of AE's, we conduct one other small experiment. We will train an AE with the same architecture as before, but with the size of the latent vector being 2, instead of 16. Then, we pass the inputs through the encoder, and look at the latent-vector encoding in the vector space. This encoding has been visualized in Figure 2.5 with two dimensions of $\mathbf{z}$ being plotted along the x and y axes. The colours correspond to the different digits, as shown in the scale on the right. One can notice that the encoding occupies a small portion of the space. Also, the encoding is not continuous in the space at all. Therefore, a random point chosen in this space, very likely does not correspond to any meaning encoding at all. This issue is addressed in Variational Auto-Encoders.

2.4 Variational Auto-Encoder(VAE)

VAE's address the shortcomings of standard AE's by implicitly forcing the latent vector $\mathbf{z}$ to behave like an i.i.d., standard gaussian vector [14]. The encoder, instead of outputting a d_z length vector, it outputs a vector of length $2d_z$. The first d_z correspond to the means μ of independent gaussian random variables, and the last d_z correspond to the

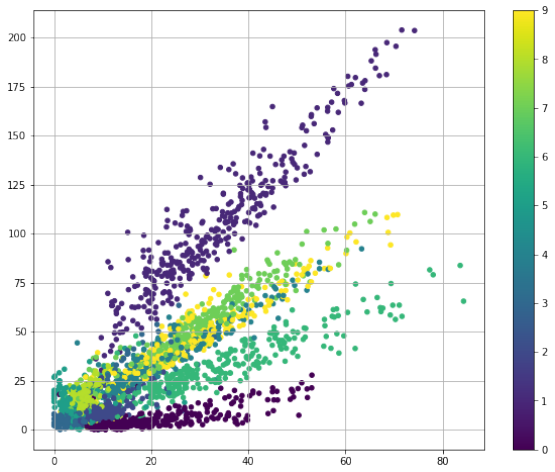


Figure 2.5: AE Latent Space Encoding

variances $\sigma^{\mathbf{2}}$. Then, the vector $\mathbf{z}$ is sampled according to $z_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$. This vector is now passed as an input to the decoder f_d. This architecture has been visualized in Figure 2.6. For the training, the parameters are forced to resemble a standard gaussian vector by modifying the loss function in (2.1) to:

$$\min_f \mathbb{E}\left[L(f(X), Y) + \lambda \sum_{i=1}^{d_z} D_{KL}\left(\mathcal{N}(0,1)||\mathcal{N}(\mu_i, \sigma_i^2)\right)\right], \tag{2.3}$$

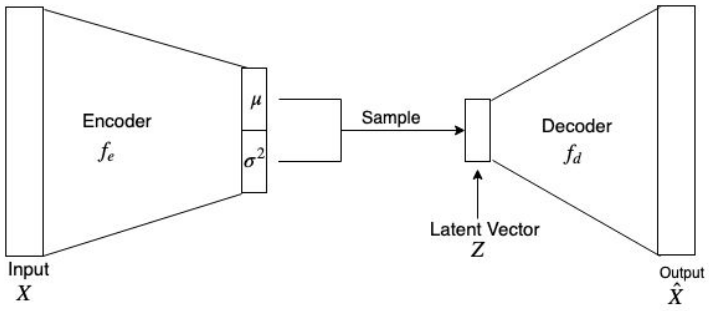
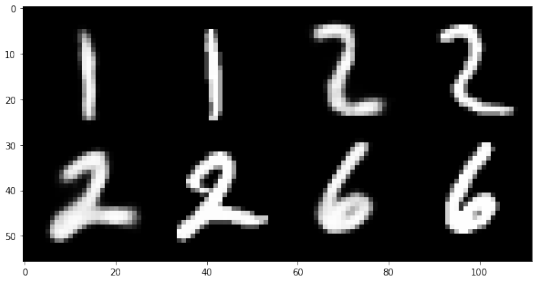
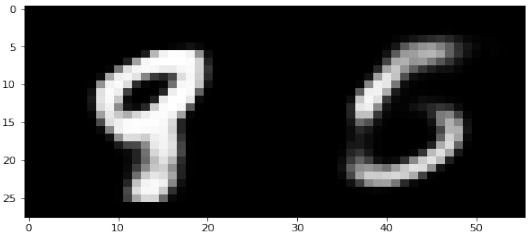


Figure 2.6: An Example VAE

where $D_{KL}(\cdot||\cdot)$ refers to the KL-divergence or relative entropy, and λ is a hyper-parameter that allows one to penalize the two losses differently. Having described the VAE model, lets check it effectiveness on the MNIST dataset.



(a) Mapping between input(on the left) and the output produced by VAE(on the right).



(b) Images generated by VAE from random latent vector

Figure 2.7: VAE on MNIST Results.

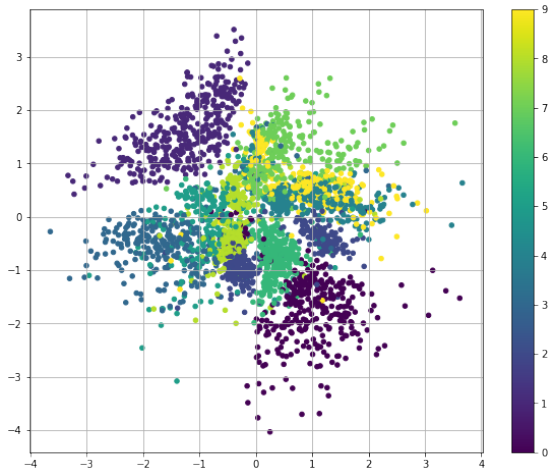


Figure 2.8: VAE Latent Space Encoding

2.4.1 Experiment on MNIST

We run the same experiment here as we did with AE's in subsection 2.3.1[3]. The reconstruction test results are shown in Figure 2.7. One can see that the mapping $\mathbf{x} \mapsto \hat{\mathbf{x}}$ is similar in quality to that produced by the AE in Figure 2.4. Additionally, the images generated by a random latent-vector loosely resemble actual MNIST digits.

The intuition behind the success of VAE as a generative model can be understood from the Figure 2.8. Notice that the points overall resemble a 2-D gaussian distribution. Most importantly, the points seem to occupy a continuous subspace. Therefore, if a random point were to be sampled from the 2-D space according to the gaussian distribution, then it is highly likely that it produces an image that resembles an MNIST digit.

[3]Source Code was modified and used from: `https://github.com/shaohua0116/VAE-Tensorflow`

CHAPTER 3

Background: Approximate Message Passing

Approximate Message Passing(AMP) is a statistical algorithm that is used to perform inference tasks on data. It was first invented to solve the problem of compressed sensing, and thus we will discuss it briefly in the next section. A while later, AMP was also shown to work with more general linear inference models. This generalization will be of use to us, and therefore we will describe it briefly in a later section.

3.1 AMP for Compressed Sensing

Compressed Sensing refers to the problem of recovering a sparse vector from its under-sampled measurements. Let $\mathbf{x} \in \mathbb{R}^N$ be a sparse vector, and let $A \in \mathbb{R}^{n \times N}$ be a measurement matrix(where $n < N$) with the entries being $A_{i,j} \sim \mathcal{N}(0, 1/n)$, i.i.d. The measurement made is $\mathbf{y} = A\mathbf{x}$. The compressed sensing problem is defined as follows:

$$\begin{aligned} \underset{\hat{\mathbf{x}}}{\text{minimize}} \quad & ||\hat{\mathbf{x}}||_1 \\ \text{subject to} \quad & A\hat{\mathbf{x}} = \mathbf{y}. \end{aligned} \tag{3.1}$$

The above optimization problem is convex, and thus can be solved by convex solvers. However, for many real life applications where the size of the vector $\mathbf{x}$ is very large, the convex-solvers are not fast enough. On the other hand, AMP is an algorithm which solves this problem with the same performance as the convex-solvers, and runs much faster [7]. It is motivated by a message-passing algorithm on dense graphs. We will very briefly describe the motivation behind the algorithm here. For more details, refer [7].

Consider the problem of finding the marginals of the following distribution:

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{i=1}^{N} \exp\left(-\beta |s_i|\right) \prod_{a=1}^{n} \delta_{y_a = (A\mathbf{x})_a}, \tag{3.2}$$

where $\delta_{y_a=(A\mathbf{x})_a}$ refers to the Dirac Delta distribution. Notice that as β is made large, the mass of the above distribution concentrates around the solution of our optimization problem (3.1). Therefore, we have reduced our optimization problem to a problem of finding the marginal of a distribution. Also notice that the distribution in (3.2) has a nice factor-graph structure with the one type of nodes being formed by the variables $\mathbf{x}_i$, and the other formed by the Dirac Delta functions(For more information on factor graphs, refer [15]). This factor graph representation lets us perform the marginalization by the messaging passing algorithm:

$$v_{i\to a}^{t+1}(x_i) \cong e^{-\beta|x_i|}\prod_{b\neq a}\hat{v}_{b\to i}^{t}(x_i), \tag{3.3}$$

$$\hat{v}_{a\to i}^{t}(x_i) \cong \int\prod_{j\neq i}v_{j\to a}^{t}(x_i)\delta_{y_a=(A\mathbf{x})_a}, \tag{3.4}$$

$$\tag{3.5}$$

where $\cong$ denotes equality up to a normalization constant. Assuming that A has i.i.d., gaussian entries, in [16] they show that the above message passing algorithm can be equivalently written as:

$$x_{i\to a}^{t+1} = \eta_t\left(\sum_{b\neq a}A_{bi}z_{b\to i}^{t}\right),$$
$$z_{a\to i}^{t} = y_a - \sum_{j\neq i}A_{aj}x_{j\to a}^{t},$$

where $\eta_t()$ is an appropriate filter described later. The goal is to have an algorithm that is very fast. Notice that the above algorithm still requires nN messages to be computed at each iteration. However, this is not the end of the story. Notice that $x_{i\to a}$'s differ in only one term for the different a's. In [16], they make a remarkable observation that $x_{i\to a}$ can be summarized by just one number x_i with a very high accuracy. The same can be done with $z_{a\to i}$. This is accomplished by incorporating the so-called "onsager-term" motivated from Statistical Physics. This algorithm(which is the AMP algorithm)

in vectorized form can bow be written as:

$$\begin{aligned} \mathbf{x}^{t+1} &= \eta\left(A^{*}\mathbf{z}^{t}+\mathbf{x}^{t}\right), \\ \mathbf{z}^{t} &= y - A\mathbf{x}^{t} + \frac{1}{\delta}\mathbf{z}^{t-1}\langle \eta_t'\left(A^{*}\mathbf{z}^{t-1}+\mathbf{x}^{t-1}\right)\rangle. \end{aligned} \tag{3.6}$$

In the above algorithm,$\langle \eta_t'\left(A^{*}\mathbf{z}^{t-1}+\mathbf{x}^{t-1}\right)\rangle$ is the onsager term. Also, $\mathbf{z}^t$ denotes the residual with respect $\mathbf{x}^t$ at time t. $\eta_t(\cdot)$ is a filter that is decided as follows:

$$\eta_t(\mathbf{u}) = \begin{cases} (\mathbf{u}-\lambda\sigma_t) & \text{if } u \geq \lambda\sigma, \\ (\mathbf{u}+\lambda\sigma_t) & \text{if } u \leq -\lambda\sigma, \\ 0 & \text{otherwise}, \end{cases}$$

where σ_t is the MSE: $\mathbb{E}\left\{(\mathbf{x}-\mathbf{x}^t)^2\right\}$ at time t, and λ is a parameter. The MSE can be tracked by an analytical state evolution equation:

$$\sigma_{t+1}^2 = \mathbb{E}\left\{[\eta_t(X+\frac{\sigma}{\sqrt{\delta}}Z)-X]^2\right\}, \quad Z\sim\mathcal{N}(0,1), X\sim \text{histogram}(\mathbf{x}^t). \tag{3.7}$$

3.2 Generalized Approximate Message Passing(GAMP)

AMP(as introduced above) is a remarkably fast algorithm that has the same performance as the optimal but slow convex-optimization algorithm. However, it seems to be restricted to work only with the compressed sensing problem as described in (3.1). Fortunately, it turns out that a with a slight modification, it also works for a much more general linear observation models [17]. This generalized version is called as Generalized Approximate Message Passing(GAMP). The general linear model that GAMP can handle is shown in Figure 3.1. The input vector $\mathbf{x}$ is generated i.i.d., according to a distribution $p(x)$. This vector is then transformed by a matrix A to produce $\mathbf{z} = A\mathbf{x}$. $\mathbf{z}$ is then passed through a memoryless channel $p(y|z)$ to produce $\mathbf{y}_i \sim p(y|z_i)$. Now, given only the observation vector $\mathbf{y}$, GAMP reconstructs both $\mathbf{z}$ and $\mathbf{x}$.

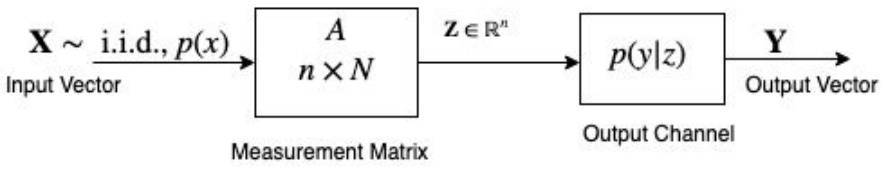


Figure 3.1: GAMP Observation Model

Note: In the literature, it is common to use the term AMP and GAMP interchangeably. We will do so as well.

The GAMP algorithm is shown in Figure 3.2. This algorithm seems to be much more complicated than the simple description of AMP in (3.6). But, upon closer look, it is possible to see that it has a very similar structure to that of AMP. Since there are two sources of randomness here(the input generation process and the output channel, as opposed to only the input generation process in AMP model), there are accordingly two filters that need to be used. Let's call them $g_{in}(\cdot)$ and $g_{out}(\cdot)$. $\hat{p}_i(t)$ and $\hat{r}_j(t)$ are the residuals, and the estimates for $\hat{z}_i(t)$ and $\hat{x}_j(t+1)$ are produced by filtering the residuals by $g_{out}(\cdot)$ and $g_{in}(\cdot)$ respectively.

Algorithm 1 Generalized AMP (GAMP)

Given a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, system inputs and outputs $\mathbf{q}$ and $\mathbf{y}$ and scalar estimation functions $g_{\rm in}(\cdot)$, and $g_{\rm in}(\cdot)$, generate a sequence of estimates $\widehat{\mathbf{x}}(t)$, $\widehat{\mathbf{z}}(t)$, for $t = 0, 1, \ldots$ through the following recursions:

1) *Initialization:* Set $t = 0$ and set $\widehat{x}_j(t)$ and $\tau_j^x(t)$ to some initial values.
2) *Output linear step:* For each i, compute:

$$\tau_i^p(t) = \sum_j |a_{ij}|^2 \tau_j^x(t) \quad (5a)$$

$$\widehat{p}_i(t) = \sum_j a_{ij}\widehat{x}_j(t) - \tau_i^p(t)\widehat{s}_i(t-1), \quad (5b)$$

$$\widehat{z}_i(t) = \sum_j a_{ij}\widehat{x}_j(t) \quad (5c)$$

where initially, we take $\widehat{s}(-1) = 0$.

3) *Output nonlinear step:* For each i,

$$\widehat{s}_i(t) = g_{\rm out}(t, \widehat{p}_i(t), y_i, \tau_i^p(t)) \quad (6a)$$

$$\tau_i^s(t) = -\frac{\partial}{\partial \widehat{p}} g_{\rm out}(t, \widehat{p}_i(t), y_i, \tau_i^p(t)). \quad (6b)$$

4) *Input linear step:* For each j,

$$\tau_j^r(t) = \left[\sum_i |a_{ij}|^2 \tau_i^s(t)\right]^{-1} \quad (7a)$$

$$\widehat{r}_j(t) = \widehat{x}_j(t) + \tau_j^r(t) \sum_i a_{ij}\widehat{s}_i(t). \quad (7b)$$

5) *Input nonlinear step:* For each j,

$$\widehat{x}_j(t+1) = g_{\rm in}(t, \widehat{r}_j(t), q_j, \tau_j^r(t)) \quad (8a)$$

$$\tau_j^x(t+1) = \tau_j^r(t) \frac{\partial}{\partial \widehat{r}} g_{\rm in}(t, \widehat{r}_j(t), q_j, \tau_j^r(t)). \quad (8b)$$

Then increment $t = t + 1$ and return to step 2 until a sufficient number of iterations have been performed.

Figure 3.2: GAMP Algorithm

The expressions for the filters, for the version of the GAMP that we will use, are:

$$g_{in}(\hat{r}, q, \tau^r) = \mathbb{E}\left[X|\hat{R} = \hat{r}, Q = q\right],$$
$$g_{out}(\hat{p}, y, \tau^p) = \frac{1}{\tau^p}(\hat{z}_0 - \hat{p}), \quad \hat{z}_0 = \mathbb{E}(z|\hat{p}, y, \tau^p).$$

Much like AMP, when $A_{ij} \sim i.i.d., \mathcal{N}(0, 1/n)$, GAMP also has a scalar state evolution equation that describes its performance through the iterations. We will not describe that here(a curious reader can refer [17]). The key take-away from GAMP for our work is the GAMP model shown in Figure 3.1. We will explore an interesting application of this on neural networks in the next 2 chapters[1].

[1]Source code was modified and used from: `https://sourceforge.net/projects/gampmatlab/`

CHAPTER 4

AMP on Synthetic Neural Network

As described in the previous chapter, GAMP has provable convergence guarantees when the transformation matrix has i.i.d., gaussian entries. Therefore, we first build our idea on a synthetic neural-network that abides by these rules.

4.1 Synthetic Neural Network Construction

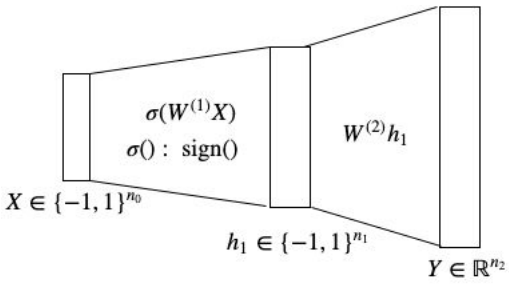


Figure 4.1: Synthetic Network Model

The synthetic neural network that we construct is shown in Figure 4.1. The input to the neural network is an i.i.d., radamacher random vector $\mathbf{X}$. It is then multiplied by the weight matrix of the first layer $W^{(1)}$ of dimension $n_1 \times n_0$, where its elements are sampled as $W_{ij}^{(1)} \sim i.i.d., \mathcal{N}(0, 1/n_1)$. The non-linearity used in the first layer is the sign function:

$$\text{sign}(x) = \begin{cases} 1 & \text{if } x \geq 0, \\ 0 & \text{if } x < 0. \end{cases}$$

The vector produced in the hidden layer will be called $\mathbf{h_1}$, and will be of size n_1. $\mathbf{h_1}$ is then multiplied by the weight matrix $W^{(2)}$ of dimension $n_2 \times n_1$, where the entries are $W_{ij}^{(2)} \sim i.i.d., \mathcal{N}(0, 1/n_2)$. There is no non-linearity applied in the second layer. Note that in order for the AMP algorithm to work, a small noise is added to both $\mathbf{h_1}$

and $\mathbf{Y}$(this noise acts as the output channel in the AMP model). So, to summarize, the neural network is represented by the function:

$$f(x) = W^{(2)}\left(\text{sign}(W^{(1)}\mathbf{X}) + \mathbf{e_1}\right) + \mathbf{e_2}, \tag{4.1}$$

where $\mathbf{e_1} \sim i.i.d., \mathcal{N}(0, \epsilon_1)$ and $\mathbf{e_2} \sim i.i.d., \mathcal{N}(0, \epsilon_2)$, for some small numbers ϵ_1 and ϵ_2.

4.2 Inference on Synthetic Neural Network

Our goal in this section is to infer the input $\mathbf{X}$ to the neural network that produced a known output $\mathbf{Y}$. Notice that the neural network in (4.1) is a non-linear function. But, it is actually composed of linear layers followed by non-linear activations. Therefore, AMP can be applied to it layer-wise to perform the inference. We will elaborate on this in the next two sections.

4.2.1 Recover $\mathbf{h_1}$ from $\mathbf{Y}$

The part of the neural network that produces $\mathbf{Y}$ from $\mathbf{h_1}$ can be written as follows:

$$f_2(\mathbf{h_1}) = W^{(2)}\mathbf{h_1} + e_2$$

This is a linear observation model, and therefore, the GAMP model from Figure 3.1 can be used to recover $\mathbf{h_1}$ from $\mathbf{Y}$. The channel model $p(y|x)$(refer to Figure 3.1 for notation) that needs to be used here is the AWGN channel model: $p(y|z) = z + e$, with $e \sim \mathcal{N}(0, \epsilon_2)$. The transformation matrix is $A = W^{(2)}$.

We also need to determine the distribution of $\mathbf{h_1}$. Note that in the GAMP model, the input distribution needs to be i.i.d for guaranteed theoretical convergence results. Here, however, the elements of $\mathbf{h_1}$ might not be i.i.d., since they might get correlated to each other through the operations in layer 1 of the neural network. However, we still use GAMP and show experimentally that the MSE of the estimation still converges to 0. To find the distribution for $\mathbf{h_1}$, notice that distribution of $\mathbf{X}$ is symmetric about

0, and the distribution of entries of $W^{(1)}$ are also symmetric about 0. Therefore, due to the application of the sign() function on the random vector $W^{(1)}X$, whose marginal distributions are symmetric about 0, we see that the entries of $\mathbf{h_1}$ behave marginally as Radamacher random variables: $\mathbf{h_1} \sim \text{Unif}\{-1, 1\}$. The experiment was run with these settings to produce an estimate of $\mathbf{h_1}$. Let's call this estimate $\hat{\mathbf{h}}_\mathbf{1}$.

4.2.2 Recover X from $\hat{\mathbf{h}}_\mathbf{1}$

Once $\hat{\mathbf{h}}_\mathbf{1}$ has been estimated, we will consider this to be the actual hidden-layer vector that was produced, and use this to estimate $\mathbf{X}$(we will show in the experiment in the next section that the estimate $\hat{\mathbf{h}}_\mathbf{1}$ has nearly MSE=0. Therefore, this is a good assumption to make.) To do so, we need to express the operations in the first layer in terms of the GAMP model in Figure 3.1.

From the synthetic network-model, the input distribution $p(x)$ is the Radamacher distribution. The transformation matrix is $A = W^{(1)}$. We will define the output-channel to incorporate the non-linearity by defining the channel as follows:

$$p(y|z) = \text{sign}(z) + e_1,$$

where $e_1 \sim \mathcal{N}(0, \epsilon_1)$. This description of the first layer of our synthetic network abides by the rules of the GAMP model. Thus, we use the GAMP algorithm to estimate $\mathbf{X}$ from it.

4.2.3 Experiment

We now run the experiment as described in the previous two sections. The parameters for the neural-network architecture chosen are

$$n_0 = 20,$$
$$n_1 = 400,$$
$$n_2 = 784.$$

The reason we choose the above network-architecture is because we will use the same architecture in the next chapter and work on a real neural network.

Upon running the experiment, it was found that the average error in $\hat{\mathbf{X}}$ was 0: $MSE = 0$. The MSE was found by running the experiment 100 times and then averaging the squared-error across the 100 experiments.

An example $\mathbf{X}$ vector and the corresponding $\hat{\mathbf{X}}$ estimated is shown in Figure 4.2.

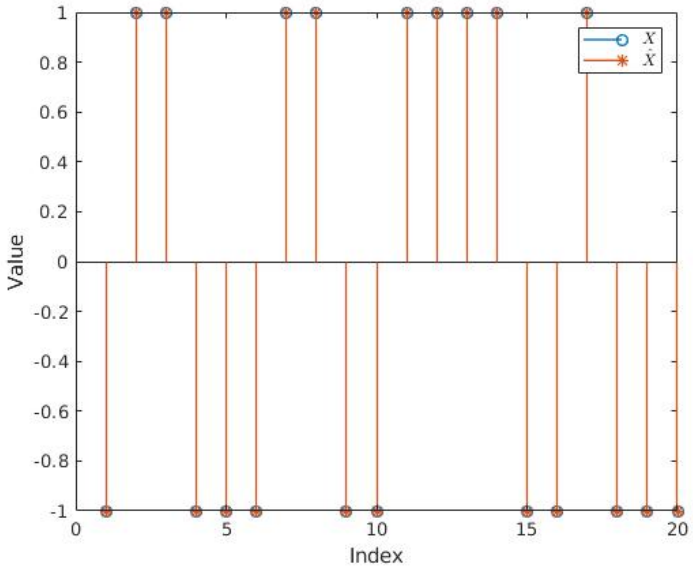


Figure 4.2: This figure shows an example run of GAMP on the synthetic network. It can be seen that GAMP has exactly estimated the values of $\mathbf{X}$.

We also plot MSE vs iteration to understand the time complexity of the algorithm in Figure 4.3. We can see that for the chosen setting of the neural network architecture, the convergence happens within 5 iterations.

Now that we have verified that GAMP works as described in the previous two sections. In the next chapter we apply GAMP to a real neural network, and illustrate a few interesting applications.

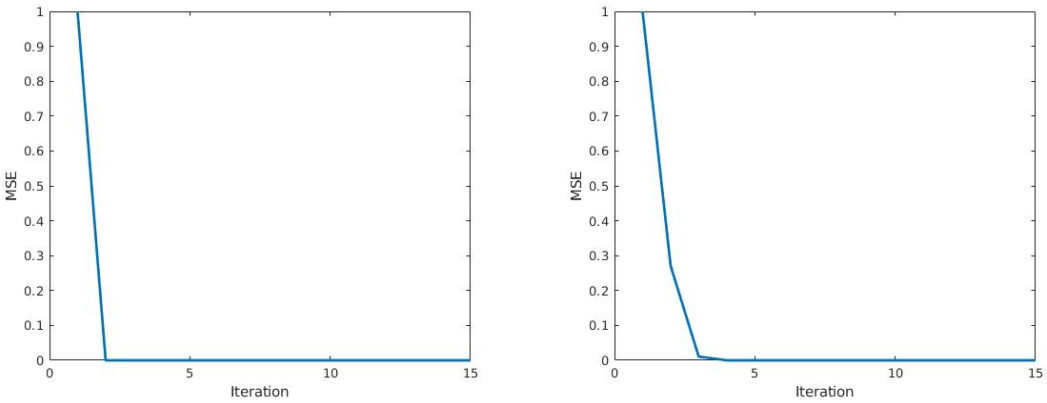


Figure 4.3: AE on MNIST Results.

CHAPTER 5

AMP on Real Neural Network

5.0.1 Neural Network Construction

We are interested in applying GAMP to the decoder part of the Variational Auto-Encoders(VAE's) which were explained in chapter 2. The VAE architecture that we use is shown in Figure 5.1. The network architecture that we use is 784-400-20-400-784. The VAE was trained on the MNIST dataset [12]. Details about the dataset can be found in subsection 2.3.1. The Adam optimizer was used, with default parameter settings on Tensorflow.

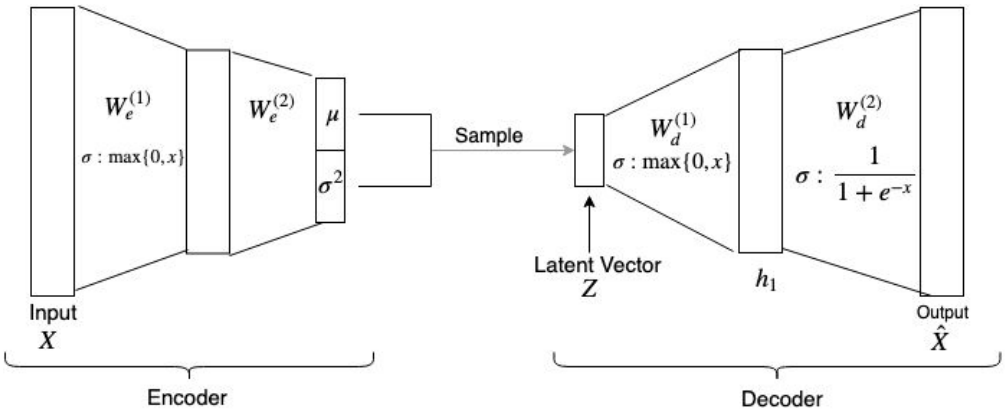


Figure 5.1: VAE Architecture. The network architecture used in the experiments will be 784-400-20-400-784.

5.1 Inference using GAMP

Once the training is done, we discard the encoder network, and only work with the decoder network. As illustrated in Figure 2.7, if a random gaussian vector with mean 0 and variance 1 is input as the latent vector to the decoder, it produces an image that looks like an MNIST digit. Now we ask the converse question: Given an MNIST digit image, can we find the latent-vector that would produce that MNIST digit as an output from the network? One can immediately see that this is the exact task that was performed in

chapter 4. However, since we are now working with a network that was trained on data, we face the following challenges:

1. Do the weight matrix entries behave like i.i.d., gaussian random variables with 0 mean?
2. Does there exist a latent vector encoding that would actually produce the given MNIST digit?
3. Do the hidden layer activations behave like i.i.d., random variables? How do we estimate their distribution?

We will talk about challenges 2 and 3 briefly, and elaborate more on challenge 1.

Challenge 2: Notice that in chapter 4, the vector $\mathbf{Y}$(on which inference was done) was actually produced by the neural network. Therefore, we knew that there exists an input $\mathbf{X}$ which would produce that $\mathbf{Y}$ exactly as an output from the network. Here, however, it could be possible that the MNIST digit that is given to us, has no latent-vector encoding that produces the exact digit as the output of our decoder network. We sidestep this challenge with a simple assumption: We assume that the representation power of our network is high enough such that there exists a latent-vector that produces as an output an image that looks quite similar to the MNIST digit given. Our experiments in the next section verify that this assumption is justified.

Challenge 3: The hidden layer activations $\mathbf{h_1}$ are certainly not independent, because of the correlations introduced by the weight matrix $W_d^{(1)}$. This same problem existed in out synthetic NN case in chapter 4 as well. Therefore, here as well, we hope that the GAMP works even though the elements of $\mathbf{h_1}$ are not i.i.d. On the other hand, estimating the distribution of $\mathbf{h_1}$ was much easier in chapter 4 because of the symmetry involved. Here, the non-linear ReLU function(that is $\max 0, x$) makes it harder to compute the distribution of h_1 directly. So, we compute it by simulation. That is, we passed 1000 random latent-vectors(with the gaussian distribution), and to produce 1000 samples of the vector $\mathbf{h_1}$. We then simply set the histogram of the samples of $\mathbf{h_1}$ produced, as the probability distribution of $\mathbf{h_1}$.

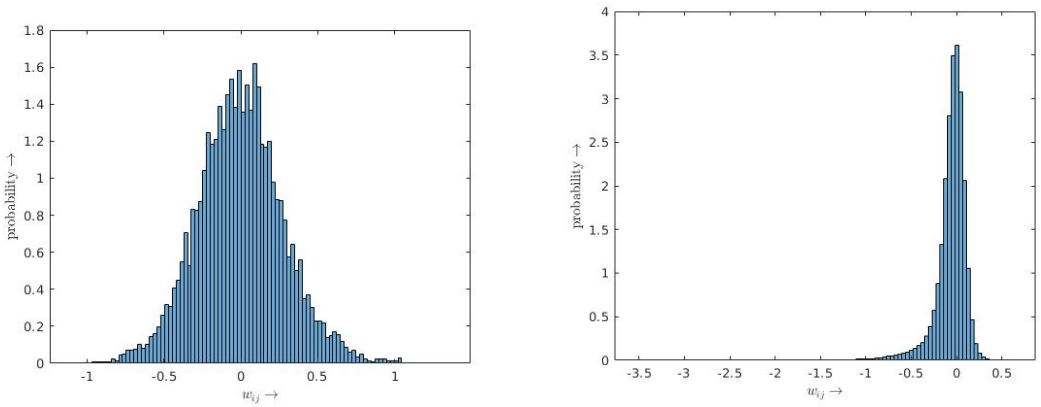


Figure 5.2: Histogram of Weight Matrices of the Decoder of VAE. The figure on the left is the histogram of the elements of $W_d^{(1)}$. The figure on the right is the histogram of the elements of $W_d^{(2)}$

5.1.1 Adjustment to the Weight Matrices

As mentioned earlier, GAMP has theoretically guaranteed convergence guarantees when the transformation matrices have i.i.d., gaussian entries. So, we start by taking a look at the histograms of $W^{(1)}$ and $W^{(2)}$ which are shown in Figure 5.2.

One can see that the elements of $W_d^{(1)}$(figure on the left) have a histogram that has 0 mean, and looks like the gaussian distribution. Therefore, we do need to make any changes to it.

On the other hand, the histogram of elements of $W_d^{(2)}$ clearly has a non-zero mean, and does not look like a gaussian distribution. To deal with this, we employ a trick. First, recall that the linear transformation in the second layer is $W_d^{(2)}\mathbf{h_1}$. Our trick is to randomly choose half of the columns in $W_d^{(2)}$, and then invert their signs(i.e., multiply them by -1). Let's call the new matrix $W_2^{(2)\prime}$. Then, we also invert the signs of the corresponding elements of $\mathbf{h_1}$, and call the new vector $\mathbf{h_1'}$. Notice that this operation ensures $W_d^{(2)}\mathbf{h_1} = W_d^{(2)\prime}\mathbf{h_1'}$. The logic behind this trick is that the histogram of $W_d^{(2)}$ was skewed towards negative numbers(as see in the right-hand figure in Figure 5.2), and this operation will now make the histogram symmetric about 0. The histogram of $W_d^{(2)\prime}$ is shown in Figure 5.3. One can now see that the histogram now has 0 mean, and slightly resembles a gaussian distribution. We thus perform GAMP with using this adjustment to the neural network.

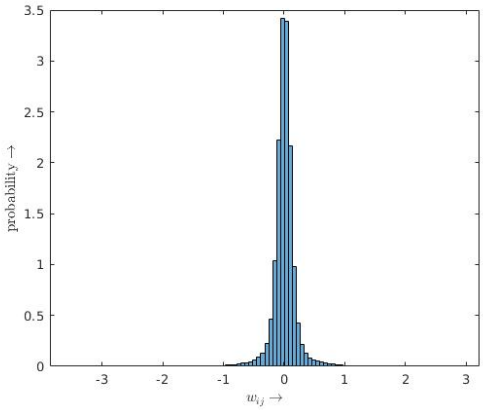


Figure 5.3: Histogram of modified weight matrix $W_d^{(2)\prime}$.

5.1.2 Inference Experiment

We ran the experiment where the output was given to be an MNIST digit, and the GAMP algorithm was used to estimate the latent-vector in the VAE that could have produced it. The network architecture and the inference algorithm were as they have been described in the previous two sections. Once the latent-vector $\mathbf{Z}$ was found, that was passed through the network to observe the output image that it produced. Some of the results are shown in Figure 5.4.

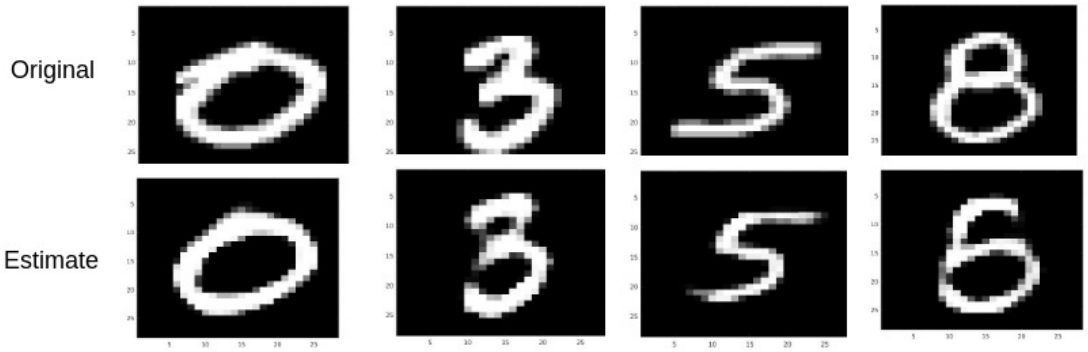


Figure 5.4: Results from estimation using GAMP. The left column consists of the MNIST digits that were considered. The images in the right column are the outputs that were produced after passing the corresponding estimated latent vectors through the decoder network of the VAE.

One can notice that the estimations do not match exactly with the original MNIST images(especially with the number 8, where the estimate looks like the number 6.) This is what was highlighted in 2nd challenge point in chapter 4. To re-emphasize, this is because the representation power of our neural network is not strong enough that each MNIST digit has an exact corresponding latent-vector. Therefore, this is not a short-

coming of GAMP itself, but a shortcoming of the power of the neural network.

5.2 Applications

Having established that the GAMP algorithm works on a real neural network, we will now illustrate some very interesting applications of this algorithm. Note that similar applications were illustrated for the Multilayered-AMP and Multilayered-VAMP, where as no similar experiments exist on AMP [**?**, **?**]. However, we show here that AMP with the modifications proposed above also performs well on the MNIST dataset.

5.2.1 Application 1: Occlusion Removal

In this application, we consider the problem of recovering the complete MNIST digit, after observing an occluded form of the digit. For an example, the middle row in Figure 5.6 shows some sample occluded digits. To be specific, we delete 6 to 8 rows in the center of the MNIST image to produce the observation $\mathbf{Y}$. Then, the task is to recover the whole MNIST image from this observation. The model is illustrated in Figure 5.5. This illustrates the change in the neural network model. The image(expressed as a vector), has a contiguous set of rows removed(shown by the black bar.) The right-hand side of the figure shows the change to be made in the $W^{(2)}$ matrix. The rows of the matrix that would be used to produce the occluded rows of the output, are removed.

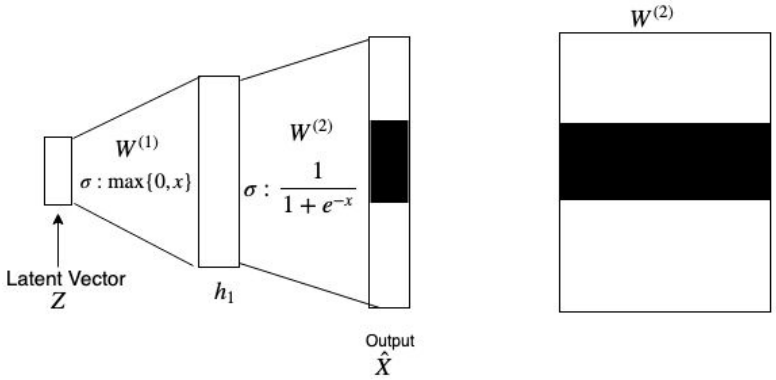


Figure 5.5: Occlusion Removal task model.

With this compressed neural network model, we now run GAMP to estimate the latent vector model that produces the observation $\mathbf{Y}$. Let's call the latent vector $\hat{\mathbf{Z}}$.

Now, this latent vector $\hat{\mathbf{Z}}$ is passed through the complete neural network to produce an estimate MNIST image. Some of the results are shown in Figure 5.6. We can see that all the images except the number 1 have been reconstructed very well(which has been estimated to look like the number 6). We can see that, the number 5 example is hard, as the occluded image could be interpreted as the number 9 as well(but, the algorithm still estimates it correctly!).

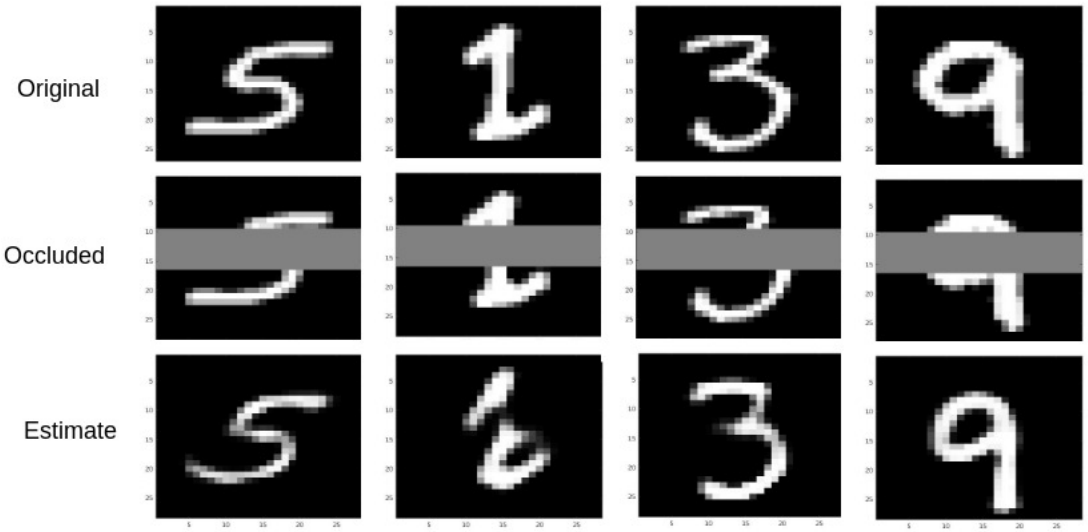


Figure 5.6: Image Estimation given Occluded Image Observation.

5.2.2 Application 2: Scattered Inpainting

In this case as well, a set of points from the MNIST image are removed. However, unlike in the previous section where one single block was removed, here a random set of points from the image are removed. For an example, see the middle of row of Figure 5.10. The model is shown in Figure 5.7. The construction of this model is very similar to that done in the Occlusion model in the previous section, where the rows of $W^{(2)}$ corresponding to the deleted points in the MNIST image are removed.

We now run GAMP on this model to estimate the MNIST image. Although this model is very similar to that of the Occlusion Model in the previous section, we observe that the dynamics of GAMP on this is different. This is illustrated in Figure 5.8. Double layer estimation refers to the algorithm described and used before, where the latent vector is estimated, and that is used to produce the output estimate. Now, observe that when estimating the latent vector $\mathbf{Z}$, we first find an estimate for the hidden layer activation $\mathbf{h_1}$. This estimate of $\mathbf{h_1}$ could be directly passed through the second layer of

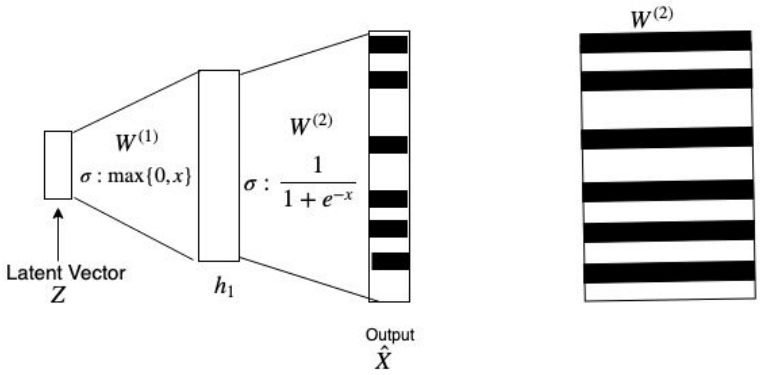


Figure 5.7: Scattered Inpainting Task model.

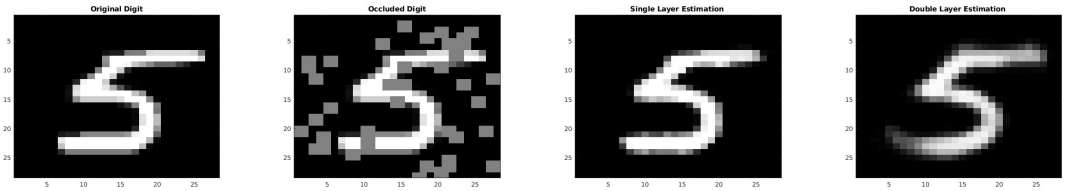


Figure 5.8: Dynamics of GAMP on Scattered Inpainting Task. Single layer estimation refers to the case where the output is produced directly from the estimate of the hidden layer activation $\mathbf{h_1}$. Double layer estimation refers to the case where the latent-vector is estimated, and that is passed through the network to produce the output.

the network to produce an estimate of the output. This algorithm is called single layer estimation.

To illustrate the difference in dynamics, we run the same experiment on the Occlusion model case of the previous section. These results are shown in Figure 5.9. Here, in the occluded task, one can see that the single layer estimation produces a poor estimate: it over-fits(i.e., tries to exactly fit) the region of the image that it sees, however the fit in the occluded section of the image is extremely poor. This is expected because single-layer estimation is performing a linear-inverse estimation, and thus only cares about fitting the points that it has observed. The double layer estimation, on the other hand, does much better here. It finds a latent-space encoding(i.e., the latent vector), which is close to what would have produced the complete image. Therefore, it produces the image of the same digit as that of the occluded image.

On the other hand, in the scattered inpainting task, the deletions are sparse and scattered. Therefore, a poor-fit in these scattered sections of the image do not affect the quality of the estimated image much. On the other hand, since the single layer

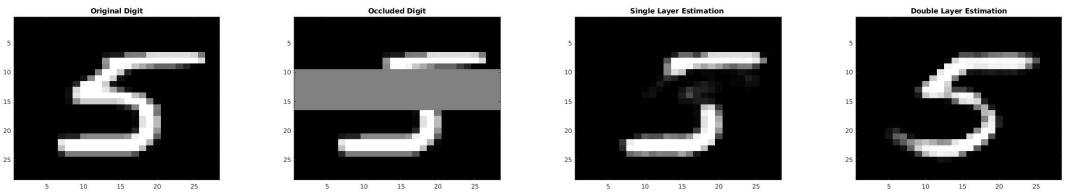


Figure 5.9: Dynamics of GAMP on Occlusion Task.

estimation produces an output that fits the non-deleted portions of the image very well, this leads to a higher quality of output.

The results of the GAMP algorithm on the scattered inpainting task are shown in Figure 5.10. The results shown in this figure are those produced from single-layer estimation.

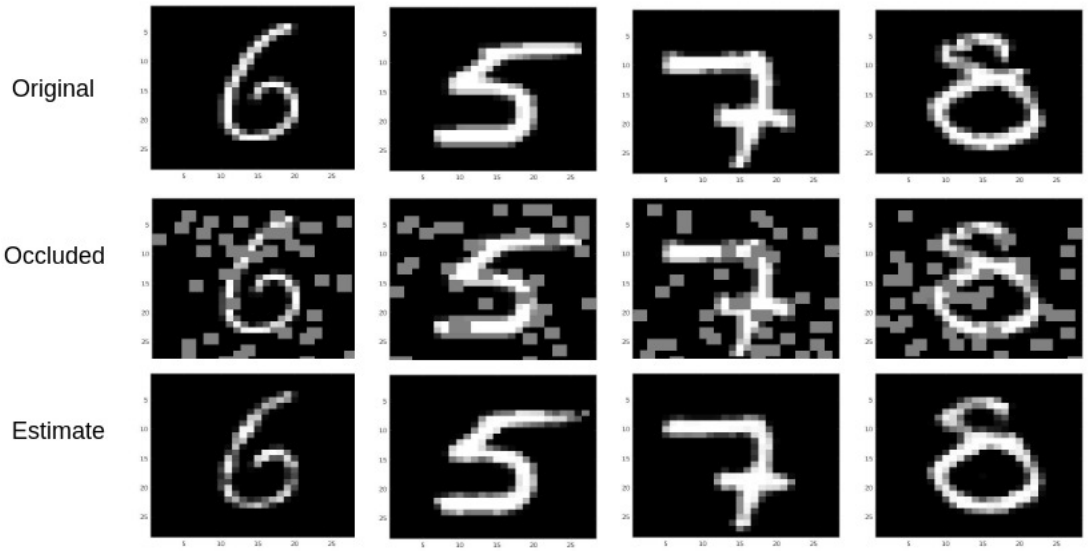


Figure 5.10: Image Estimation given scattered missing points.

5.2.3 Application 3: Noise Removal

For the last application, we consider the task where gaussian noise(the SNR corresponding the noise was varied from 1 to 10) is added to an image and the task is to remove the noise from the image. In this case, we do not make any changes to the neural network model. We simply run GAMP on the neural network to find the latent vector corresponding to the noisy observation $\mathbf{Y}$. And then, we pass the latent vector through the network to produce the denoised image.

The results are shown in Figure 5.11. As one can see, the noise that we have added is

very high, to the point that it is difficult for humans to reconstruct the digit. The GAMP algorithm seems to be able to identify the digit and reconstruct an approximately similar looking image, in most cases. In the example of number 5 in the figure however, the algorithm seems to detected it as the number 3.

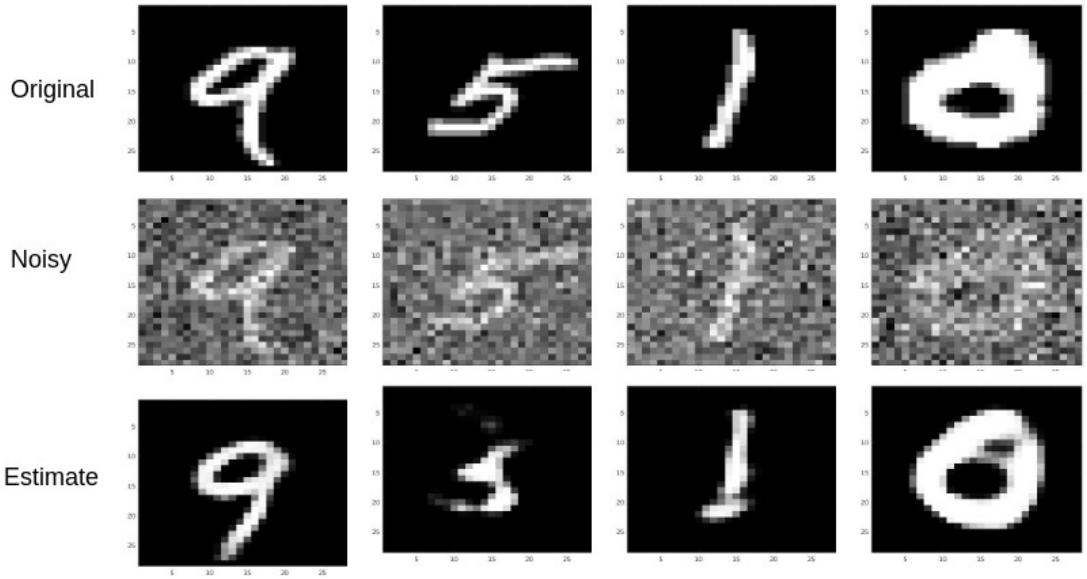


Figure 5.11: Image Estimation given Noisy Observation.

CHAPTER 6

Conclusion and Future Work

In this work, we illustrated an interesting application of GAMP in image-estimation problems using neural networks. Although GAMP only has theoretical convergence guarantees when the observation matrix A has i.i.d., gaussian entries, and i.i.d., input generation process, we show here that GAMP still works in practical cases where these conditions do not hold.

One line of future work could involve finding a theoretical explanation for why GAMP works very well for the models that we considered. For another line of work, notice that we only illustrated how to use GAMP with feed-forward fully-connected neural networks. One could probably also try to use GAMP with other neural network models such as convolutional neural networks and recurrent neural networks. Doing so will open up the doors for many more interesting applications on much more wider and complicated set of data.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[2] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, pp. 91–99, 2015.

[3] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, pp. 30–42, Jan 2012.

[4] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[5] L. Fridman, D. E. Brown, M. Glazer, W. Angell, S. Dodd, B. Jenik, J. Terwilliger, J. Kindelsberger, L. Ding, S. Seaman, H. Abraham, A. Mehler, A. Sipperley, A. Pettinato, B. Seppelt, L. Angell, B. Mehler, and B. Reimer, "MIT autonomous vehicle technology study: Large-scale deep learning based analysis of driver behavior and interaction with automation," *CoRR*, vol. abs/1711.06976, 2017.

[6] D. L. Donoho *et al.*, "Compressed sensing," *IEEE Transactions on information theory*, vol. 52, no. 4, pp. 1289–1306, 2006.

[7] D. L. Donoho, A. Maleki, and A. Montanari, "Message-passing algorithms for compressed sensing," *Proceedings of the National Academy of Sciences*, vol. 106, no. 45, pp. 18914–18919, 2009.

[8] J. Rick Chang, C.-L. Li, B. Poczos, B. Vijaya Kumar, and A. C. Sankaranarayanan, "One network to solve them all–solving linear inverse problems using deep projection models," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 5888–5897, 2017.

[9] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[10] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[11] D. H. Ballard, "Modular learning in neural networks," in *Proceedings of the Sixth National Conference on Artificial Intelligence - Volume 1*, AAAI'87, pp. 279–284, AAAI Press, 1987.

[12] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010.

[13] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[14] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[15] T. Richardson and R. Urbanke, *Modern coding theory*. Cambridge university press, 2008.

[16] D. L. Donoho, A. Maleki, and A. Montanari, "Message passing algorithms for compressed sensing: I. motivation and construction," in *2010 IEEE Information Theory Workshop on Information Theory (ITW 2010, Cairo)*, pp. 1–5, IEEE, 2010.

[17] S. Rangan, "Generalized approximate message passing for estimation with random linear mixing," in *2011 IEEE International Symposium on Information Theory Proceedings*, pp. 2168–2172, IEEE, 2011.