Making a Neural Network learn to say Don't Know

Laltu Roy

M. Tech Computer Science

Supervisor: Prof. Nikhil R. Pal
ECSU
Indian Statistical Institute
Kolkata

This project is submitted as a dissertation for the requirements of the Master of Technology programme in Computer Science at Indian Statistical Institute.

Acknowledgement

I would like to thank my dissertation supervisor, Dr. Nikhil R. Pal, Professor, Electronics and Communication Sciences Unit, Indian Statistical Institute, Kolkata, for agreeing to guide me and helping me to undertake work in the topic. Without his continuous guide and support, this would not have been possible.

I am also very much thankful to Suvro Da, Suchismita Di, Manish, Arnob for helping me though out the project with their valuable time and suggestions and thus I was able to give this work a final form.

Laltu Roy
MTech(CS), 2nd year
Roll No.- CS1813
ISI, Kolkata
July, 2020

Abstract

Classification problem is a popular topic within machine learning community. One of the major problem in classification task is how to handle incoming patterns which are unusual and different in some measure. For such novel input the classifier should be able to distinguish them as unknown type and don't make any decision. In this work we try to address that problem.

Contents

1 Introduction

Autoencoders are a specific type of feed-forward neural networks. In autoencoder, the input is the same as the output. They compress the input into a lower-dimensional *code* and then reconstruct the output from this representation [7, 8].

An autoencoder consists of 3 components: *encoder, code and decoder*. The encoder compresses the input and produces the code, the decoder then reconstructs the input from this code.

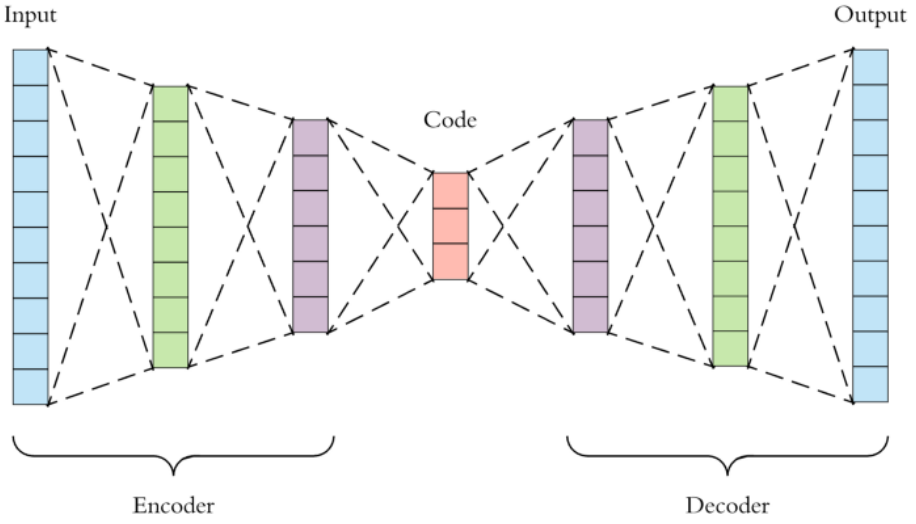

Figure 1: Schematic diagram of Autoencoder[8].

First the input passes through the encoder, which is a fully-connected Neural Network, to produce the code. The decoder, which has the similar structure, then produces the output only using the code. The goal is to get an output identical with the input [8].

The output of autoencoder will not be exactly the same as the input, it will be a close but degraded representation.

In 2014 *Ian J. Goodfellow* [1] introduced Generative Adversarial Networks(GANs), which are a powerful class of neural networks.

There are mainly two network in GAN architecture, called Generator and Dis-

criminator. The generator(G), which aims to generate realistic data, while the discriminator(D), tries to discriminate real data from the data generated by G.

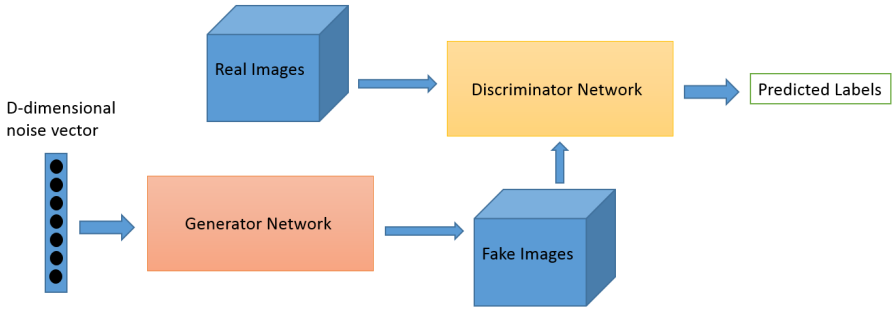

Figure 2: Schematic diagram of GAN [11].

1.1 Training of GANs

Training of GANs is based on a two-player game between the two network. The Generator try to make fake samples similar to real samples from a latent space and fed to the Discriminator. The input of the Discriminator is either the real data or output of Generator. The Discriminator try to separate these fake samples from the real ones. Depending upon this discriminator's output GAN trains itself through back-propagation.

In the training process, The discriminator outputs a value D(x) indicating the chance that x is a real image. Our objective is to be maximize the chance to recognize real images as real and generated images as fake. i.e. the maximum likelihood of the observed data. The objective function for Discriminator is:

$$\max_{\theta_D} V(D) = \mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))] \tag{1}$$

The Generator wants to maximize the value D(G(z)). Its objective function is:

$$\min_{\theta_G} V(G) = \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))] \tag{2}$$

Combining equation 1 and 2, we get the objective function of GAN as:

$$\min_{\theta_G} \max_{\theta_D} V(G, D) = \mathbb{E}_{x \sim p_{data}}[\log D(x)] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))] \quad (3)$$

The two network play a **min-max** game [4]. The discriminator tries to maximize the objective function $V(G, D)$ by only controlling it's parameter θ_D. On the other hand, the generator tries to minimize the objective function $V(G, D)$ by only controlling it's parameter θ_G.

2 Novelty Detection

Novelty detection is the task of classifying test data that differ in some respect from the data that are available during training. Novel data refers to the new, unusual observations which do not occur regularly or simply different from the observations encountered in training time. In training time the network does not learn anything about novel data [12].

In machine learning systems, not all possibilities can be inputted during training, so there will always be new kinds of data and possibilities that will arise after training period.

One of the basic requirements from a good classifier is generalization - the ability to classify input data that is reasonably similar to the training data. Usually there are no requirements on how the classifier should behave for new types of input that differ substantially from the data that are available during training. For such novel input the classifier will produce erroneous output and classify it as one of the available classes. Ideally, we would like that the classifier, in addition to its generalization ability, be able to detect novel inputs, or in other words, we would like the classifier to say, **"Don't Know"** [10].

For one class problem, if new data points are unusual or far from the training data we can call them novel data. But for a multi-class problem, if the new data points are unusual or far from training points or not so close to any class we don't put them to any class. we assign them to a new class called **"Don't Know"** class. To do that we will add one extra class label to the classifier.

3 Proposed Approach

The proposed framework is similar to Generative Adversarial Networks and closely related to Sabokrou's work [6] on one-class novelty detection method. The framework is consist of two components: (1) Network E and (2) Network D. Network E works as a generator/re-constructor of data points, and Network D works as a Discriminator. In this framework, instead of a generator used in GAN [1] models, an Autoencoder is considered as network E.

3.1 Theory

Suppose we are given data $\{x_i\}_{i=1}^{n}$, where $x_i \in X$ from real data distribution P_t. To learn the data distribution properly it is not necessary to estimate density of P_t. Instead, the network D trains an autoencoder E to help it to learn the distribution better. E generate/construct samples from $\tilde{x}_i = (x_i \sim P_t) + (\nu \sim N(0, \sigma))$, into $E(\tilde{x}_i) = x_i' \sim P_\theta$, such that $P_\theta \approx P_t$. ν is noise sampled from normal distribution $N(0, \sigma)$ with standard deviation σ. Noise is added to make network E robust to noise.

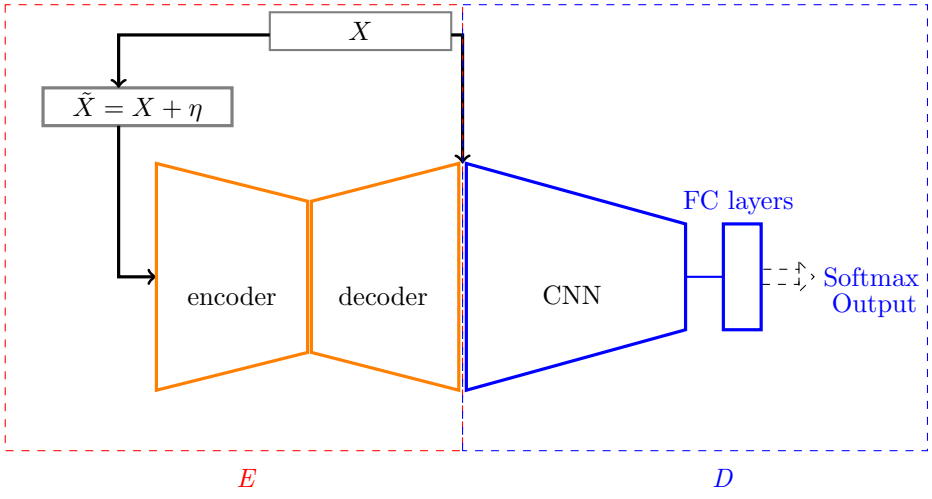

Figure 3: Overview of the structure [6].

D measure the similarity between P_θ and P_t via samples $\{x_i\}_{i=1}^n$ and $\{x'_i\}_{i=1}^n$ during the training to understand P_t better. The autoencoder aims to generate samples close to P_t.

A Bad generator(week and unbalanced) [3] is considered for learning a good Discriminator. Similar to GANs training network E and D play a min-max game to optimize the following objective function,

$$\min_E \max_D = (\mathbb{E}_{x\sim p_t}[\log(D(x))] + \mathbb{E}_{\tilde{x}\sim p_t+N(0,\sigma)}[\log(1-D(E(\tilde{x})))]) \tag{4}$$

To Reduce the linear separability between generated samples and real samples, we used *Feature Matching* [5] which is equivalent to linear maximum mean discrepancy [9] employing linear first moment matching in the space of discriminator features. Feature matching loss is calculated as follows,

$$FM = ||\mathbb{E}_{X\sim p_t}[f(X)] - \mathbb{E}_{\tilde{X}\sim p_t+N(0,\sigma)}[f(E(\tilde{X}))]|| \tag{5}$$

where function f is the representation of input data in an intermediate layer of network D. We considered the second last layer representation for calculating *feature matching loss*.

To train the network we calculate the loss $\mathbb{L}_{E+D}$ as the loss function of joint network E+D. Now the whole network is trained with loss function,

$$\mathbb{L}_{E+D} + \lambda FM \tag{6}$$

where $\lambda(>0)$ is trade-of hyper-parameter between two loss functions for relative importance.

3.2 Algorithm

We can see from the theory that *feature Matching* is the distance/dissimilarity between real data and generated data representation in feature space of discriminator. We try to generate/construct samples with similar representation of generated data to train discriminator learn the data distribution more accurately. We try to do that by adding *feature matching* as regularizer to the generator in while training. So, our algorithm looks like below,

Algorithm:

for number of iteration **do**

- Sample minibatch of m samples $\{x_1, x_2, \cdots x_m\}$ from real dataset
- Sample minibatch of m samples $\{\tilde{x}_1, \tilde{x}_2, \cdots \tilde{x}_m\}$ from noisy dataset
- **Get** output $\{x'_1, x'_2, \cdots x'_m\}$ of $\{\tilde{x}_1, \tilde{x}_2, \cdots \tilde{x}_m\}$ from E
- Update D by ascending it's stochastic gradient :

$$\nabla \frac{1}{m} \sum_{i=1}^{m} [\log D(x_i) + \log(1 - D(E(x'_i)))]$$

- **Get** intermediate representation of $X = \{x_1, x_2, \cdots x_m\}$ and $\tilde{X}' = \{x'_1, x'_2, \cdots x'_m\}$ from Discriminator D
- Calculate feature matching loss $FM(X, \tilde{X}')$
- Update E by descending it's stochastic gradient with added regularizer term:

$$\nabla \frac{1}{m} \sum_{i=1}^{m} [\log(1 - D(E(x'_i)))] + \mu FM(X, \tilde{X}')$$

end for

4 Experiment

4.1 Architecture

Network E: An Autoencoder is used as Network E. We used Convolutional Neural Network(CNN) in both encoder and decoder of the Autoencoder.

Figure given below shows the architecture of network E.

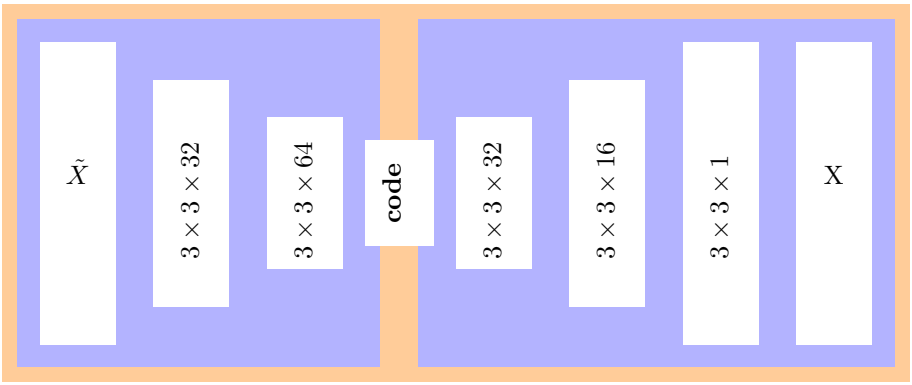

Figure 4: Network E architecture. the tuple inside a box indicate kernel size(row $\times$ column) $\times$ output channels.

Network D: The architecture of network D consist of several convolution layers followed by fully connected layers.

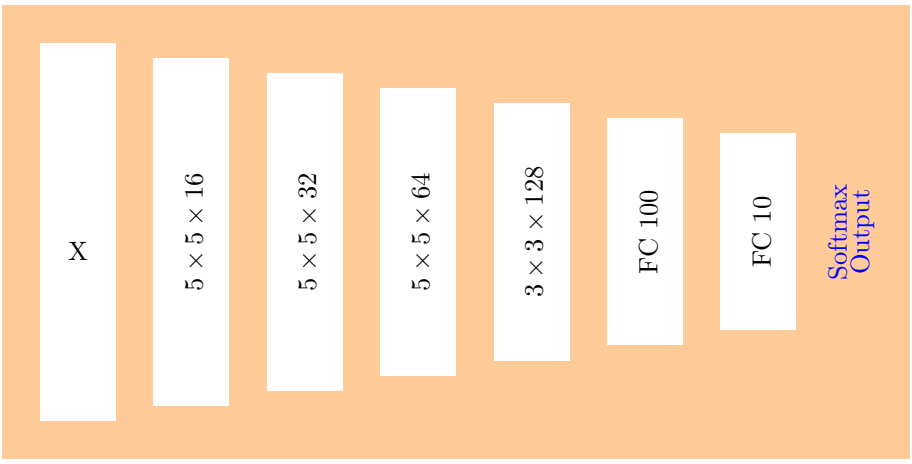

Figure 5: Network D architecture.

After each convolution layer we used Batch Normalization, and LeakyRelu is used as activation function. We didn't used MaxPooling after convolution layers.■

We trained our model on **MNIST** dataset which contains 28*28, single channel, 60,000 training images. The model is implemented using Tensorflow Framework 1.x and Python 3.x and run on **Google Colab**.

Hyper-parameter: Gaussian Noise is added to input of network E with standard deviation 0.07. We used Adam optimizer with learning rate=0.002. And epsilon of BatchNormalization is set to 0.001. The size of minibatch is set to 128 and number of epochs is set to 10.

4.2 Result

MNIST dataset consist of 60,000 handwritten digits from '0' to '9'. Each of the ten categories of digits is taken as novel data. During training no novel data is taken, only the other nine digits are considered.

We test the **MNIST** test dataset two ways, one is considering the index of the maximum value of the output vector as the class of input data. Another is using threshold on max value of the output vector. The threshold value is set to 0.5. The table given below shows some results of the Experiment, the digits in the first row, is considered as the novel data for that experiment.

	0	2	7	5	8	9
without thresold	86±3.0	83±2.5	87±2.5	85±3.0	86±2.5	82±2.0
with thresold	87±2.5	83.5±3.0	87±2.5	85±3.0	88±2.0	81±2.5

4.3 Result analysis

From the above result we can see that the results varies between this two approach by small percentage, but in second approach it detects more test data correctly than the first one.

5 Conclusion

In this dissertation, we have proposed a general framework for multi-class classification and novelty detection in images. Our architecture consists of two modules, generator/reconstructor and discriminator. The former helps latter to learn the concept of real target classes images. After training the model, D learns to classify real images to their respective classes and also detect unknown data which are different from real ones in some measure.

Although our model is not performing as the level of state-of-the-art methods, it is performing quite good with such simple algorithm.

References

[1] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio *Generative Adversarial Networks* https://arxiv.org/pdf/1406.2661.pdf

[2] Nikhil R. Pal, Bikram Karmakar *How to make a neural network say "Don't know"* https://www.sciencedirect.com/science/article/pii/S0020025517311234

[3] Zihang Dai, Zhiling Yang, Fan Yang, William W. Cohen, Ruslan Salakhutdinov *Good Semi-supervised Learning That Requires a Bad GAN* https://arxiv.org/abs/1705.09783

[4] Ian J. Goodfellow *NIPS 2016 Tutorial: Generative Adversarial Networks* https://arxiv.org/pdf/1701.00160.pdf

[5] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen *Improved Techniques for Training GANs* https://arxiv.org/abs/1606.03498

[6] Mohammad Sabokrou, Mohammad Khalooei, Mahmood Fathy, Ehsan Adeli *Adversarially Learned One-Class Classifier for Novelty Detection* https://arxiv.org/pdf/1802.09088.pdf

[7] Jake Krajewski *Autoencoder neural networks: what and how?* Article

[8] Arden Dertat *Applied Deep Learning - Part 3: Autoencoders* Article

[9] Arthur Gretton, Karsten M. Borgwardt†, Malte J. Rasch, Bernhard Scholkopf, Alexander Smola *A Kernel Two-Sample Test* https://pdfs.semanticscholar.org/9748/432d69e7da4b935338b303d3c7468997db12.pdf

[10] Mark Kliger, Shachar Fleishman *Novelty Detection with GAN* https://arxiv.org/pdf/1802.10560.pdf

[11] Chris Nicholson *A Beginner's Guide to Generative Adversarial Networks (GANs)* Article

[12] *Novelty Detection* Article