Generative Adversarial Networks using Probabilistic Principal Component Analysis

Group 4
Hardil Mehta (1401018), Chintan Saraviya (1401026),
Kashish Shah (1401048), Rupande Shastri (1401102), Kishan Raval (1401117)

***Abstract*—The need for generative models has grown with the increasing complexity of neural networks. Currently the leading implementation of these generative models are Generative Adversarial Networks (GANs) as they provide relatively good representations of the training samples fed to it. However, training them is a notoriously slow and complex process. In this paper principal components obtained through probabilistic principal component analysis are provided in place of actual full rank data. This makes the training process faster and reduces the effect of corruption in the training data.**

I. Introduction

In the field of deep learning there is a monotonic increase in the complexity of neural networks in the wake of the race for optimum accuracy. As the complexity of neural networks increases, so does the need for more labelled samples to train them. Even with the tremendous amount of information generated in the real world, there seems to be a shortage of available training data for these complex neural networks. Since training samples are limited in the real world, there is a growing interest in models that can learn to generate data similar to the training samples we need. These models form a separate category of machine learning models called Generative Models.
There are two categories of models in Machine Learning:

- Discriminative Models: It models model a function that maps input data x to output labels y by learning the conditional distribution $P(x|y)$.
- Generative Models: The focus is on learning the joint distribution $P(x, y)$. This is used for generating more (x, y) pairs in a similar range and can be converted to $P(x|y)$ for classification.

Apart from the generative ability, generative models have another advantage over discriminative models as they can learn the structure of unlabelled data x as well, which helps in semi-supervised and unsupervised forms of learning.

II. Generative Adversarial Networks

Of the many implementations of generative models Generative Adversarial Networks (GANs), GANs arguably fare better than others in terms of accuracy of training sample representation. It is a direct implementation that uses latent code and has no good way of quantification.
The idea is to pit two neural networks against each other in order to achieve an optimum state. A GAN consists of a generative model (or a generator) and a discriminative model (discriminator). The generator takes noise as input and generates samples. These samples are fed into the discriminator along with original training samples to be distinguished as 'real' or 'fake' depending on their similarity with the original training samples. The output of the discriminator is fed to the generator so that it generates images closer to 'real'. This cycle goes on until a state is reached where the samples created by the generator are always similar enough to the original data so the discriminator always confirms the samples as 'real'.

III. Probabilistic Principal Component Analysis

The process of training a GAN is notoriously slow and complex. This gets worse when the training samples are of high dimensions. Moreover, if the training set is noisy or corrupted, the corruption passes on to the generated images as well. A solution to counter these issues is to use the principal components of the training samples, use them in the discriminator so that the generator generates more similar principal components and then construct new training samples from the generated principal components.
For a given d-dimensional vector of observed data $\mathbf{t}$, the relationship between $\mathbf{t}$ and a corresponding q-dimensional vector of latent variables $\mathbf{x}$ is given as $\mathbf{t} = \mathbf{W}\mathbf{x} + \mu + \epsilon$.

The computation of $\mathbf{W}$ gives us the coefficients of the dimensionality-reduced matrix.

We convert the latent variable model into a probabilistic model and obtain a mean μ and covariance matrix $\mathbf{S}$ of the data via Maximum Likelihood estimation or log-likelihood method in PPCA. Expectation Maximization uses the probabilistic model to compute the parameter $\mathbf{W}$ by the following steps:

- **E-step:** Compute the posterior probability $\mathbf{x}|\mathbf{t}$
- **M-step:** Using the posterior, choose $\mathbf{C}$ and $\mathbf{R}$ to maximize joint likelihood of $\mathbf{t},\mathbf{x}$

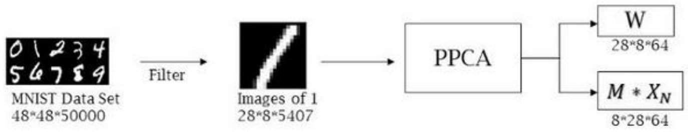
IV. Implementation

We have implemented our model using the MNIST dataset. The dataset contains images of handwritten digits from zero to nine, of the same size (28 x 28). The result of our implementation should be images similar to the MNIST images. Our implementation involves the following steps:

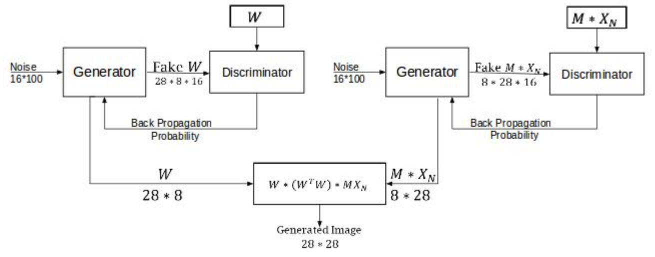
1) Extract principal components of MNIST training images using the PPCA-EM algorithm.
2) Using these principal components as training samples, train the GAN to generate more principal components.

3) Construct images using the generated principal components and compare them with the original training samples.

Another method for comparison of the generated samples with the original is by finding the norm of the difference between the generate samples and the original samples and comparing it to a threshold value. The figure below shows the pre-processing (extracting principal components) process -



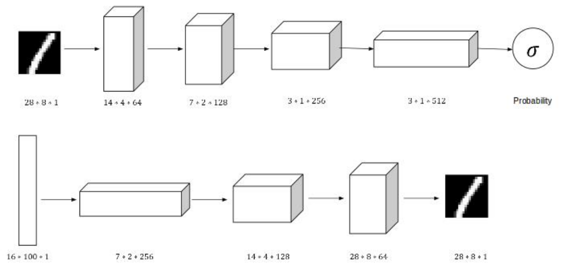
We filtered the MNIST dataset to take out all the images of 'ones'. Extracting principal component analysis of the images, we end up with matrices W and $M * Xn$. W contains the coefficients of the dimensionality reduced matrix and $M * Xn$ gives us the components in the new dimensions. The GAN model is given in blocks below -



In our GAN, we have two generator-discriminator pairs working together to generate new values for W and $M * Xn$. These new values are used to construct new images by the formula

$$recovered\ data = W * (W^T * W)^{-1} * M * Xn$$

The discriminator uses convolution to classify while the generator uses deconvolution to generate samples. The process in our case is shown below -



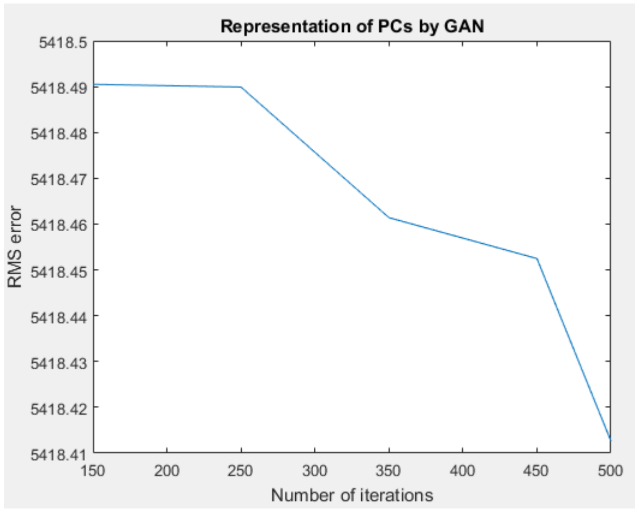
An example of reconstructed images is shown in the figure below, beside an original sample -

TABLE I. ROOT MEAN SQUARE ERRORS WITH RESPECT TO ORIGINAL SAMPLE

Iteration	Image	W	$M * Xn$
150	59.7589016891894	677960.479127273	5418.49049591214
250	65.6402971215527	677960.456930971	5418.48990601572
350	186.180851819857	677960.473354998	5418.46140040418
450	2890.21755082488	677960.474512392	5418.45246940186
500	563.810006801496	677960.474981213	5418.41263485471

On comparing with the original sample, we get results as seen in the table.

Comparing generated principal components with the original, we find that the difference reduces with number of iterations as shown in the figure.



V. CONCLUSION AND FUTURE WORK

The GAN implemented did not give a good representation of the PCs, as can be seen from the results. This was partly because of shortage of training samples of PCs to train the GAN with.

Another possible approach to generate PCs with limited data is to augment all images of a class into one matrix and extract PCs of that matrix. This reduces the dimensions to a large extent making it easier to train the GAN.

REFERENCES

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, *Generative Adversarial Networks*, 2014.

[2] Atienza, Rowel. "GAN By Example Using Keras On Tensorflow Backend Towards Data Science Medium". Medium. N.p., 2017. Web. 23 Apr. 2017.

[3] Goodfellow, Ian. "Generative Models". OpenAI Blog. N.p., 2016. Web. 23 Apr. 2017.