Adversarial Autoencoders

Sagar Patni.
(shpatni@asu.edu)

I. Introduction

The paper Adversarial Autoencoder [1] discusses about "how the adversarial autoencoder can be used in applications such as semi-supervised classification, disentangling style and content of images, unsupervised clustering, dimensionality reduction and data visualization". In this project I will try to reproduce the results given in the paper for supervised, semi supervised and unsupervised learning on MNIST dataset. Further, I will test the CNN architectural changes

1. All convolutional net
2. Eliminating the Fully Connected layers
3. Batch Normalization

II. Background

In this section, we will discuss some the basic concepts which are important to understand the Adversarial Autoencoder.

A. *Autoencoder*

An autoencoder is a neural network which tries to generate the identity mapping with its input so that its output x~ is like input. Learning the identity mapping seems trivial at first glance but we can discover interesting structures about the data by placing constraints on the network such as the number of hidden units.

The autoencoder is comprised of 2 parts viz. encoder and decoder. The encoder generates the compressed representation of the input data if the number of hidden units is less than the input units. The decoder tries to regenerate the input data from the compressed vector. It learns to do so using backpropagation.

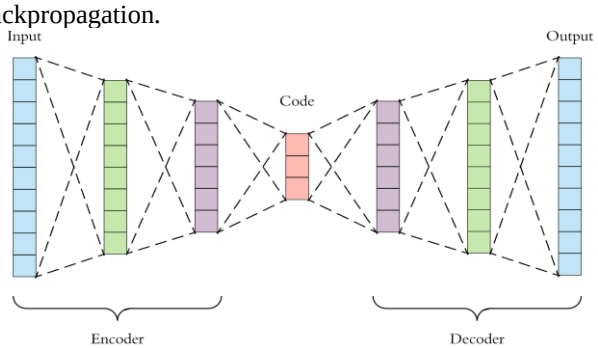

Figure 1. Autoencoder
https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

Denoising and dimensionality reduction are two main applications of the autoencoders. With proper sparsity constraints and dimensionality, autoencoder can learn data projections which are better than PCA and other basic techniques. The latent representation generated by autoencoder can be used for unsupervised tasks such as document clustering.

In natural language processing tasks autoencoders are used to generate semantic word mappings from the input corpus[5]. Recursive Autoencoder architecture is used to encoder sentences with word mapping into compressed vector. Autoencoders can be used for machine translation in which the decoder learns convert the latent representation generated by input in a source language into a target language.

B. *KL Divergence*

Kullback–Leibler divergence is a measure of how one probability distribution diverges from a second probability distribution. KL divergence 0 indicates that the two distributions are similar while 1 indicates that the two distributions are different. In deep learning, KL divergence is used along with the objective function to approximate the true distribution to selected distribution.

$$D_{\mathrm{KL}}(P\|Q) = -\sum_i P(i)\,\log\frac{Q(i)}{P(i)}$$

C. *Variational Autoencoder*

The variational autoencoders are an extension to autoencoders. In autoencoders, we can regenerate the image by passing the latent representation of image through the decoder. The autoencoders are not probabilistic in nature to generate an image from them we need to memorize the latent representation.

In case of variational autoencoders, the network is forced to represent the input vectors as a distribution. We achieve so by enforcing the latent vector to be a unit Gaussian probability distribution. To generate new images from the network we just need to sample from the unit Gaussian probability distribution and pass it through the decoder.

The loss term for variational autoencoder is the sum of generative loss which is mean squared error that measures the accuracy of image reconstruction and KL divergence that measure how closely the latent variables match unit Gaussian.

To learn the by gradient descent parameter from the network we apply a simple reparameterization trick, the hidden layer will generate the vectors for mean and standard deviation. We generate the sample using $z=\mu+\sigma\odot\epsilon$

The lower bound of objective function for VAE is given as,

$$\mathcal{L}(\theta,\phi;X) = E_{z\sim q_\phi(Z|X)}(\log P_\theta(X|Z)) - \mathcal{D}_{KL}(q_\phi(Z|X)||p_\theta(Z))$$

The first part of the equation represents reconstruction error and second represents the KL divergence between two distributions.

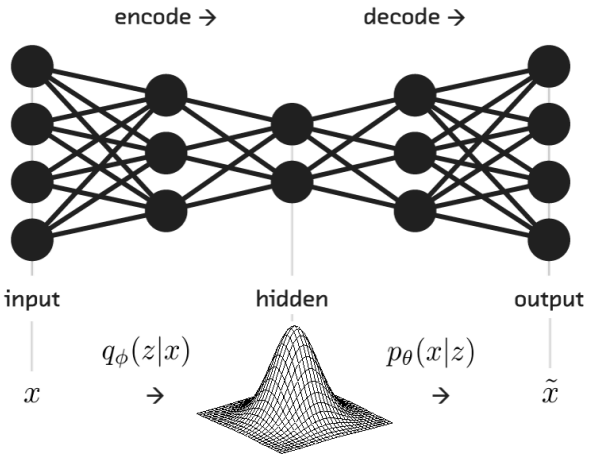

Figure 2. Variational Autoencoder
http://fastforwardlabs.github.io/blog-images/miriam/imgs_code/vae.4.png

D. GAN

In Generative Adversarial Networks, we simultaneously train two models a generative model G to capture the data distribution and a discriminator model D which estimates the probability that the sample comes from the data distribution rather than the distribution generated by G. [3] The goal of the generator is to generate images to fool the discriminator. The goal of the discriminator is to identify images coming from the generator. While doing so the generator learns to produce the images which cannot be differentiated from the real ones.

The objective function for GAN is given as,

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

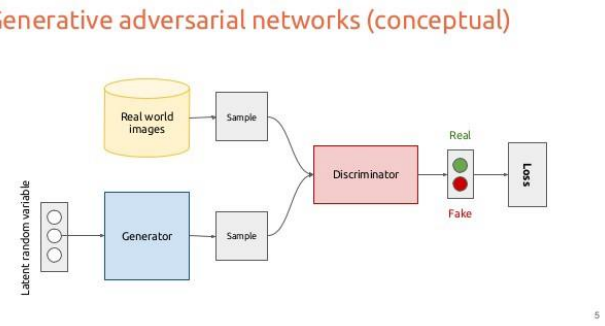

Figure 3. Architecture of GAN
https://cdn-images-1.medium.com/max/1600/0*Mwpzq1rqmc-2LJsx.

The steps GAN takes while learnings,

1. The generator is fed with a vector with random numbers.
2. The generated images from the generator are fed into discriminator along with the original image.
3. The discriminator takes both the images and returns probability that the images by the generator are authentic or not.

While training the generator we keep the discriminator output constant and vice versa. Both generator and discriminator are trying to optimize an opposing optimization functions. Any side of GAN can overpower other if the discriminator is too good it will return the values 0 and 1 which makes generator learning difficult and if the generator is too good it will discriminator will cease to learn. Different strategies and learning rates are adopted to learn the GAN effectively.

III. ADVERSARIAL AUTONENCODER

The images in the Generative adversarial network are generated from random noise vector of which we cannot keep track. We can compare generated images directly to the originals, which is not possible when using a GAN.

A downside to the VAE is that it uses direct mean squared error instead of an adversarial network, so the network tends to produce more blurry images. [reference]

The paper tries to narrow this gaps by combining GAN and VAE. AAE uses similar loss function as VAE, instead using KL divergence to impose prior distribution on hidden code vector we use adversarial training to achieve it.

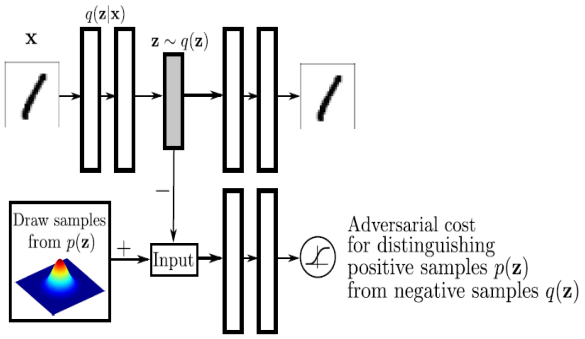

Figure 4. Adversarial Autoencoder [1]

Figure 4 shows the architecture of the Adversarial Autoencoder. Top portion represents a simple autoencoder which tries to reconstruct the input image by latent vector q(z|x). The bottom portion of the image represents adversarial network which tries to determine whether the sample came from latent distribution q(z) or sampling distribution p(z) The training procedure imposes distribution p(z) to latent distribution q(z).

The network is trained with SGD in two phases: 1. reconstruction phase 2. regularization phase. In reconstruction phase the autoencoder tries to minimize the reconstruction error from the input. In regularization phase the discriminator sets apart the true samples from fake ones and then it updates the generator.

Where the label 'y' are one hot encoded vectors. We add extra label which represents that the label is no present this can be used during test time.

In this setting, we examine different machine learning approaches 1. Supervised Learning 2. Semi-Supervised Learning 3. Unsupervised Learning.

IV. SUPERVISED AUTOENCODER

In supervised setting, we will disentangle the label information from the style information for that we incorporate label information 'y' in the adversarial training stage as shown in the fig.., so the latent distribution 'z' mainly represents the style information. The network learns to differentiate the decision boundaries based on the label information provided.

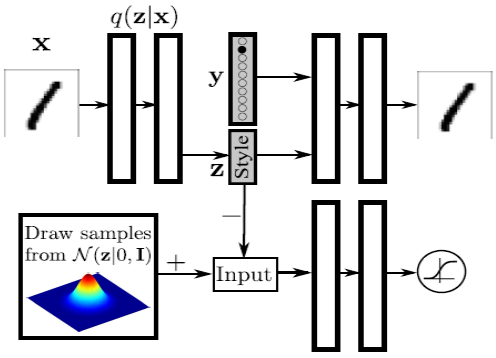

Figure 5. Supervised training with AAE [1].

To generate samples, we can sample from latent distribution z we can provide the class label information to the adversarial stage.

V. SEMI-SUPERVISED AUTOENCODER

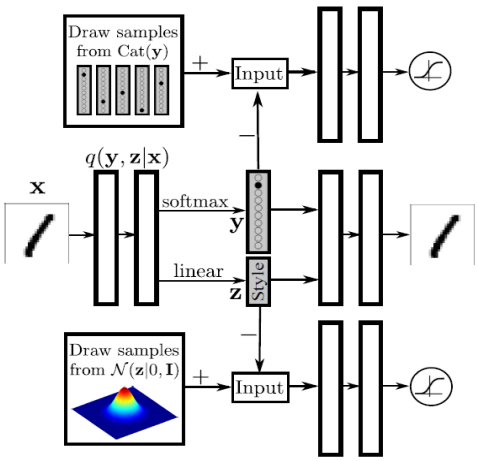

Figure 6. Semi Supervised Learning with AAE [1].

To develop models for semi-supervised learning, which will learn the labelling information to improve performance, we add one more adversarial layer which allows the network to learn the label encoding 'y'. The first adversarial network ensures that the latent variable does not include any style information by imposing a categorical distribution to it.

During training, In the reconstruction phase the network minimizes reconstruction error for the input data. In the regularization phase two discriminators distinguish between true samples and the generated samples and then it updates the generator network.

VI. UNSUPERVISED LEARNING

The architecture for the unsupervised learning is like the semi-supervised learning. In this case the adversarial network for categorical distribution draws one hot encoded sample from the number of clusters we wish to impose on network.

VII. WORK DONE SO FAR

- Understanding mathematical formulation of the GAN and VAE.
- Understand the research paper objective and results.
- Developing DCGAN and VAE autoencoder in Tensorflow to understand fundamentals of both.

VIII. REMAINING TASKS

- Develop the AAE in Tensorflow.
- Compare the results obtained for AAE with the VAE.
- Apply different CNN architectural changes.

IX. REFERENCES

[1] Adversarial Autoencoders Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, Brendan Frey
[2] Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. Alec Radford, Luke Metz, Soumith Chintala
[3] Generative Adversarial Nets Ian J. Goodfellow, Jean Pouget-Abadie_, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozairy, Aaron Courville, Yoshua Bengioz
[4] Tutorial on Variational Autoencoder. Carl Doersh.
[5] https://deeplearning4j.org/word2vec.html