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.

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.

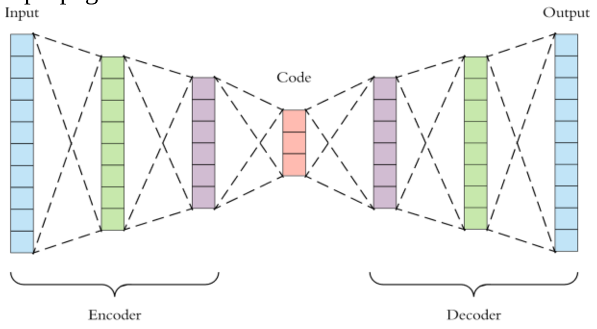


Figure 1. Autoencoder
https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

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.

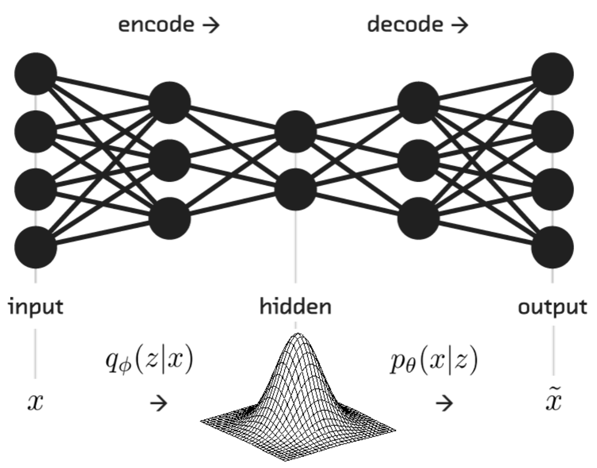


Figure 2. Variational Autoencoder
http://fastforwardlabs.github.io/blog-images/miriam/imgs_code/vae.4.png

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[3],

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

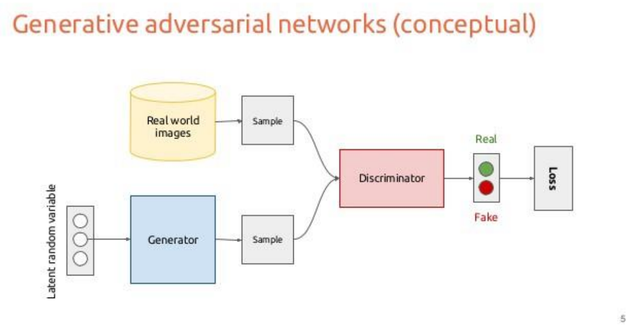


Figure 3. Architecture of GAN
https://cdn-images-1.medium.com/max/1600/0*Mwpzq1rqmc-2LJsx.

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 AUTOENCODER

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. [4]

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.

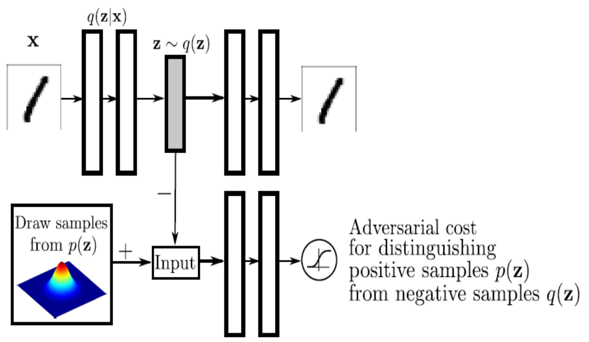

x
$q(\mathbf{z}|\mathbf{x})$
$\mathbf{z} \sim q(\mathbf{z})$
−
Draw samples from $p(\mathbf{z})$
+
Input
Adversarial cost for distinguishing positive samples $p(\mathbf{z})$ from negative samples $q(\mathbf{z})$

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 figure 5, so the latent distribution 'z' mainly represents the style information. The network learns to differentiate the decision boundaries based on the label information provided.

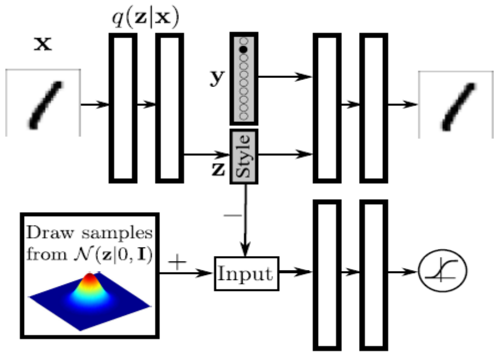

$q(\mathbf{z}|\mathbf{x})$
$\mathbf{x}$
$\mathbf{y}$
$\mathbf{z}$
Style
Draw samples from $\mathcal{N}(\mathbf{z}|0,\mathbf{I})$
+
−
Input

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.

The supervised AAE is trained with learning rate .0001 on the entire MNIST dataset for 100 epochs using minibatch gradient descent of batch size 100. The encoder has input size of 784, 2 hidden units of dimension 1000 each and 15 output units. The decoder takes input from encoder along with a categorical variable for length 10, it has 25 input units, rest part of the decoder is the exact opposite of the encoder. The discriminator takes input from a normal distribution of mean 0 and standard deviation 5 with 15 input units.

V. Semi-Supervised autoencoder

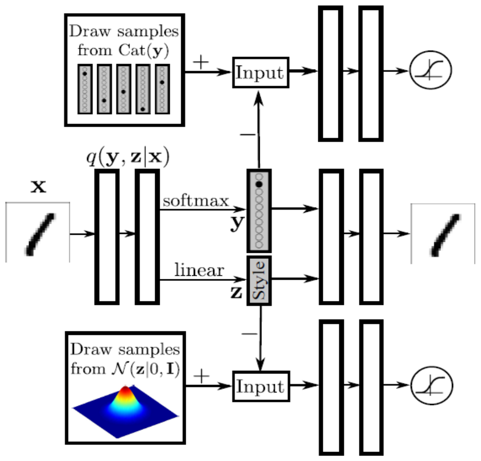

Draw samples from Cat($\mathbf{y}$)
+
Input
−
$q(\mathbf{y},\mathbf{z}|\mathbf{x})$
$\mathbf{x}$
softmax
$\mathbf{y}$
linear
$\mathbf{z}$
Style
Draw samples from $\mathcal{N}(\mathbf{z}|0,\mathbf{I})$
Input

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

To use unlabeled data to increase the classification performance that would be obtained like unlabeled data. we assume that the data is generated by latent class variable comes from the categorical distribution and gaussian distribution

$p(y) = cat(y)$
$p(z) = N(z|0,I)$

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.

The semi-supervised AAE is trained with learning rate .0001 on the entire MNIST dataset for 100 epochs using minibatch gradient descent of batch size 100. The encoder has input size of 784, 2 hidden layers of dimension 1000 each and 15 output units. The decoder takes input from encoder along with a categorical variable for length 10, it has 25 input units, rest part of the decoder is the exact opposite of the encoder. The discriminator takes input from a normal distribution of mean 0 and standard deviation 5 with 15 input units. It has 2 hidden layers of dimension 1000 each and a single output unit. The second discriminator takes the categorical variable of lengths 10 as input and it has 2 hidden layers of dimension 1000 each and a single output unit.

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. TensorFlow Implementation

For implementation of the project, I used TensorFlow deep learning framework. For supervised learning AAE was trained for 100 epochs with learning rate 0.001. The TensorFlow implementation details for loss function and optimizer are as below,

```
self.autoencoder_loss = tf.reduce_mean(tf.square(self.x_op -
self.decoder_output))

self.dc_loss_real =
tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(labels=tf.ones_lik
e(self.d_real), logits=self.d_real))
self.dc_loss_fake =
tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(labels=tf.zeros_li
ke(self.d_fake), logits=self.d_fake))

self.dc_loss = self.dc_loss_fake + self.dc_loss_real

self.generator_loss =
tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(labels=tf.ones_lik
e(self.d_fake), logits=self.d_fake))
```

During optimization phases of discriminator, we keep generator constant and for optimization phase of generator we keep discriminator constant. It can be achieved in TensorFlow by passing variable list to optimizer

```
self.autoencoder_optimizer =
tf.train.AdamOptimizer(learning_rate=SupervisedConfig.learning_rate,
beta1=SupervisedConfig.beta1).minimize(self.autoencoder_loss)

self.discriminator_optimizer =
tf.train.AdamOptimizer(learning_rate=SupervisedConfig.learning_rate,
beta1=SupervisedConfig.beta1).minimize(self.dc_loss,
var_list=self.dc_var)

self.generator_optimizer =
tf.train.AdamOptimizer(learning_rate=SupervisedConfig.learning_rate,
beta1=SupervisedConfig.beta1).minimize(self.generator_loss,
var_list=self.en_var)
```

Similarly, for semi-supervised learning, AAE was trained for 100 epochs with learning rate 0.001. The TensorFlow

```
self.autoencoder_loss = tf.reduce_mean(tf.square(self.x_op -
self.decoder_output))

self.dc_g_loss_real =
tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(labels=tf.ones
_like(self.d_g_real), logits=self.d_g_real))

self.dc_g_loss_fake =
tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(labels=tf.zeros
_like(self.d_g_fake), logits=self.d_g_fake))

self.dc_g_loss = self.dc_g_loss_fake + self.dc_g_loss_real

self.dc_c_loss_real =
tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(labels=tf.ones
_like(self.d_c_real), logits=self.d_c_real))

self.dc_c_loss_fake =
tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(labels=tf.zeros
_like(self.d_c_fake), logits=self.d_c_fake))

self.dc_c_loss = self.dc_c_loss_fake + self.dc_c_loss_real

self.g_g_loss =
tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(labels=tf.ones
_like(self.d_g_fake), logits=self.d_g_fake))

self.g_c_loss =
tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(labels=tf.ones
_like(self.d_c_fake), logits=self.d_c_fake))

self.g_loss = self.g_c_loss + self.g_g_loss

self.encoder_training_loss =
tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=self.y_i
p_l, logits=self.encoder_op_l))
```

implementation details for loss function and optimizer are as shown, in this case we train two discriminators one for categorical distribution and second for gaussian distribution.

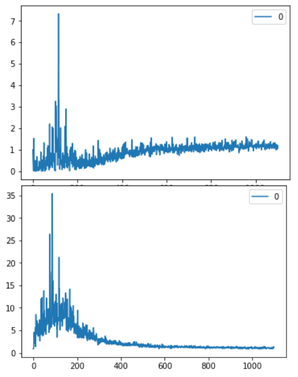
Similarly, during optimization phases of discriminator, we keep generator constant and for optimization phase of generator we keep discriminator constant. It can be achieved in TensorFlow by passing variable list to optimizer.

```
self.autoencoder_optimizer =
tf.train.AdamOptimizer(learning_rate=Config.learning_rat
e, beta1=Config.beta1).minimize(self.autoencoder_loss)

self.discriminator_g_optimizer =
tf.train.AdamOptimizer(learning_rate=Config.learning_rat
e, beta1=Config.beta1).minimize(self.dc_g_loss,
var_list=self.dc_g_var)

self.discriminator_c_optimizer =
tf.train.AdamOptimizer(learning_rate=Config.learning_rat
e, beta1=Config.beta1).minimize(self.dc_c_loss,
var_list=self.dc_c_var)

self.generator_optimizer =
tf.train.AdamOptimizer(learning_rate=Config.learning_rat
e, beta1=Config.beta1).minimize(self.g_loss,
var_list=self.en_var)

self.supervised_encoder_optimizer =
tf.train.AdamOptimizer(learning_rate=Config.learning_rat
e,beta1=Config.beta1).minimize(self.encoder_training_los
s, var_list=self.en_var)
```

VIII. EXPERIMENTAL SETUP

For this experiment, I used Intel i7 7th generation processor with 16GB ram.

IX. OBSERVATIONS



- When the training process is started the generator, loss is greater than the discriminator loss while training they fluctuate between high and low values as training progress both the values get close to each other.

Figure 7 Discriminator loss and generator loss, supervised training.

- AAE models are trained end-to-end but for semi supervised VAE models we must train it one layer at a time [6].
  The understand the rationale behind the parameters used in the paper [1]. I changed some of the hyperparameters of the network, below are the results for 10 epochs with supervised AAE

  Learning rate 0.001: While training, the generator loss is substantially greater than the discriminator loss thus the network learns slowly.

Learning rate 0.01: While training, the generator loss is less compared to the discriminator loss. The network learns quickly. Even with 10 iteration the network was able to generate good samples which are better than compared to setting given in the paper with 10 iterations. The network learned to disentangle style and generate data samples for few of the gaussian priors.

Learning rate 0.1: The images generated are blurry. With this high learning rate, the discriminator outperforms the generator which ceases the learning process, the images tend to blurry and does not give out any useful information.

- With less learning rate the network majorly focuses on reconstruction of the image, as the discriminator is commensurate to the generator in term of prediction, while slowly imposing data distribution to latent variables with adversarial training.
- Changes with number of hidden neuron (784, 512, 64): By reducing the number of neurons in the hidden layers the training speed significantly increased but the images generated in this setting were not good as generated by configuration given in the paper [1], given same number of iterations.

X. RESULTS

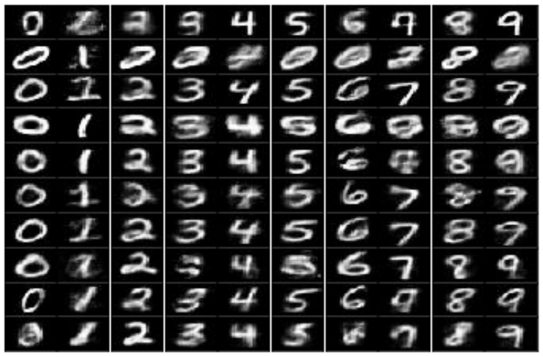


Figure 8. Supervised training with AAE (Disentangling content and style on MNIST 100 epochs)

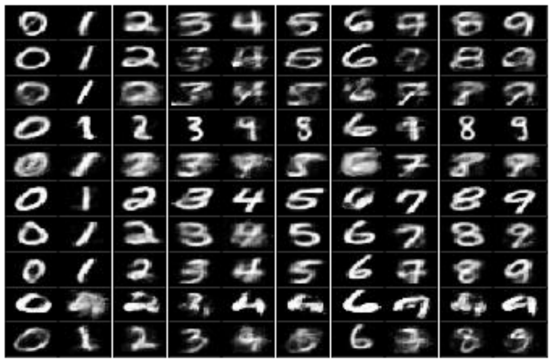


Figure 6.1 Supervised training with AAE (Disentangling content and style on MNIST)

In above figure, each row is generated with same input prior and different categorical values. We can see that the images belonging to different rows significantly differs in styles.

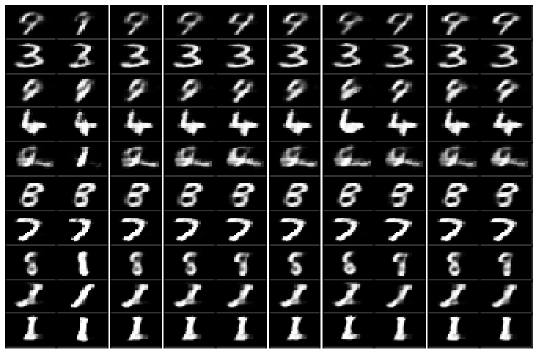


Figure 9. Semi-Supervised training with AAE (Images generated by decoder in 100 epochs)

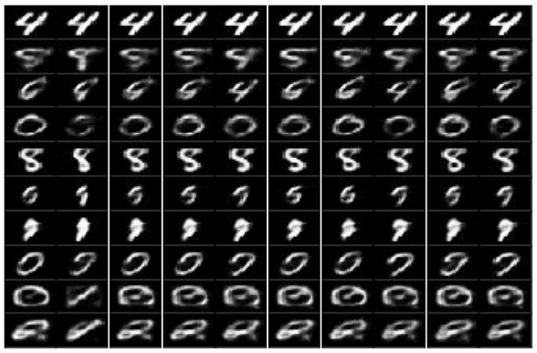
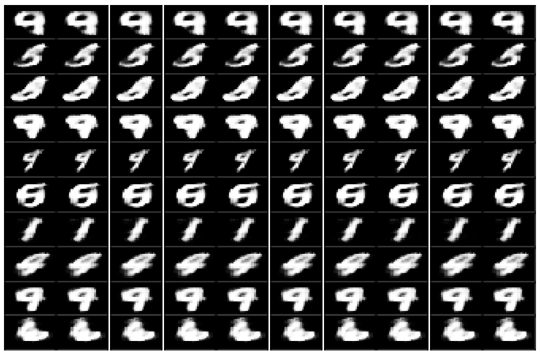


Figure 9.1 Semi-Supervised training with AAE (Images generated by decoder for semi supervised learning on MNIST)

In above figure, each row is generated with a random vector belonging to a distribution and a categorical variable. From above figure we can observe that by changing the categorical variable there are little changes in the output. But we can see that we achieve significant classification accuracy with this setup. For **encoder classification accuracy was 94.56%**.

For unsupervised learning,



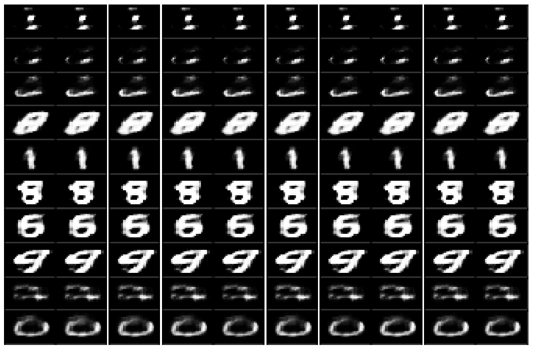


Figure 10 un-supervised training with AAE 20 clusters 10 epochs

The above figures show the result of unsupervised learning with AAE, each row represents the images generated using unsupervised learning approach. In this experiment we used 20 clusters, each row in above image represents a cluster.

XI. CONCLUSION

In this project we re-formulated the proposed adversarial training as a variational inference algorithm for discreet and continuous latent variables for probabilistic autoencoder. Further, we reproduced similar results given in the paper for supervised, semi supervised and unsupervised learning on MNIST dataset.

XII. FUTURE WORK

- The idea of adversarial autoencoder can be extended to disentangle textual information from the image.
- The setup can be tested with word2vec as categorical distribution to generate images with style fluctuations given textual data.
- The MNIST dataset is grayscale dataset, this idea can be further taken for images having varied elements.

XIII. LEARNINGS

The adversarial training can be useful to impose data distribution to neural network and its better than autoencoder architecture.

VAE vs GAN: VAE uses decoder network to impose the posterior distribution to the latent variable while GAN uses adversarial training to directly map the data distribution. With VAE we can directly compare the Images generated to the input images. But the images generated are blurry.

AAE vs. GAN and VAE: AAE is combination of GAN and VAE, it has autoencoder like VAE, but it replaces the KL divergence loss with adversarial training. AAE utilizes latent distribution to generate images instead using random vector.

The implementation details of the GAN, VAE, AAE are understood. Imposing the data distribution gives flexibility to user to generate images.

XIV. CONTRIBUTION

Individual project.

- Report: 100%
- Implementation: 100%
- Interpreting the results: 100%

XV. 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
[6] Diederik P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models.