Representation Learning: Deep Learning Approach

Phani Ram Sayapaneni
Department of Computer Science
University at Buffalo
Buffalo, NY 14226
phaniram@buffalo.edu

Abstract

Through this project, the problem of representation learning has been analyzed through recent innovations in the area of unsupervised feature learning and deep learning, covering auto-encoders, manifold learning, and deep networks. The main objective is to identify good representations for data and explore ways to compute those representations.

1 Introduction

Data representations play a key role in the performance of many machine learning algorithms. Majority of the efforts in machine learning applications go into the feature designing and representation learning. The quest for AI based solutions in every field is motivating the design of more powerful representation-learning algorithms. It is important to have these representations developed in an unsupervised fashion to quickly scale the applications and construct the models quicker. This project focuses on exploring the deep learning techniques to yield more useful representations.

2 Elements of interest for good representations

2.1 Distributed representations

Distributed representations of concepts are representations composed of many elements(neurons) that can be set separately from each other. Essentially, it's a many to many relationship, neural networks are best at this. Each neuron captures many representations and each representation can be represented by many neurons. Using just one neuron for each representation is highly inefficient, that's where the distributed learning techniques excel.

2.2 Depth and abstraction

Depth is another important aspect to representation learning. If the goal is to find best features out of the data, depth provides more abstract features at higher levels of representation (such as eyes, nose from human faces). Depth also provides re-use of features. In deep architectures, all this abstraction from feature re-use leads to more abstract features which are usually scale invariant and insensitive to specific changes/noise to the input data. This provides a huge leverage to many classification or objection detection problems.

2.3 Disentangling factors of variation

With small changes in few underlying factors present in the data, the input data distribution might change, this makes it difficult for many real time machine learning applications to stay robust to such variations, especially for sensory data. In order to learn representations which are invariant to such factors, it is important to disentangle such factors from the data in first place. Thereby building features that are insensitive to the variations(uninformative) in data for the task at hand.

3 Building Deep Representations

Regular probabilistic models, which could be used to learn latent variables by computing p(Z|X) posterior over latent variables, given input should also work. Unfortunately, these techniques become impractical or intractable due to the computational complexity associated with sampling methods, error resulting from approximate methods.

3.1 Auto- Encoder

An Auto encoder is a neural network that is trained to copy the input to it's output. Though, it is no more than an identity function, it learns interesting properties about the data, when constrained through it's architecture. Auto encoders can not only be used for dimensionality reduction, but to generate data.

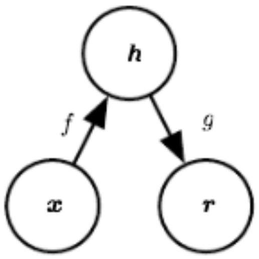


Fig. 1 Shows the graphical model of an auto encoder.

The Auto Encoder consists of 2 parts, the encoder function h = f(x) and the decoder function r = g(h). The learning process is to learn these two functions f, g such that g(f(x)) = x. Though the network can learn perfect g, f functions, the architecture imposes few constraints such that reconstruction can never be perfect.

3.2 Under complete Auto- Encoder

Mere copying the input to the output won't be much useful. Using an under complete auto encoder, we try to reduce the size of hidden layer h, such that the network learns to represent h most useful elements(salient features), to be able to reconstruct the input representation. This reveals many interesting properties about the input when we look the data through it's encoded latent space.

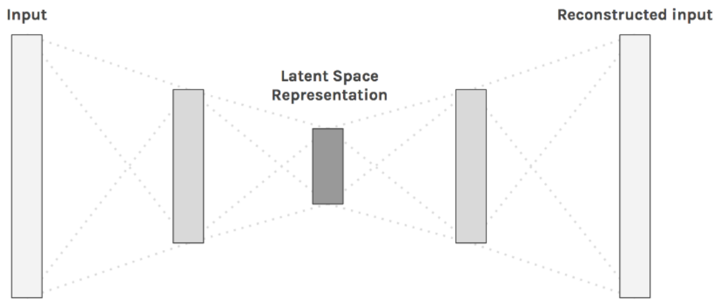


Fig. 2 Shows the architecture of an under complete auto encoder

This learning process is to minimize the occurred in reconstruction: L (g(f(x)) –x). A simple auto encoder has been implemented with 2 hidden layers, the reconstruction and the learnt weights reveal that such an auto encoder is not very useful.

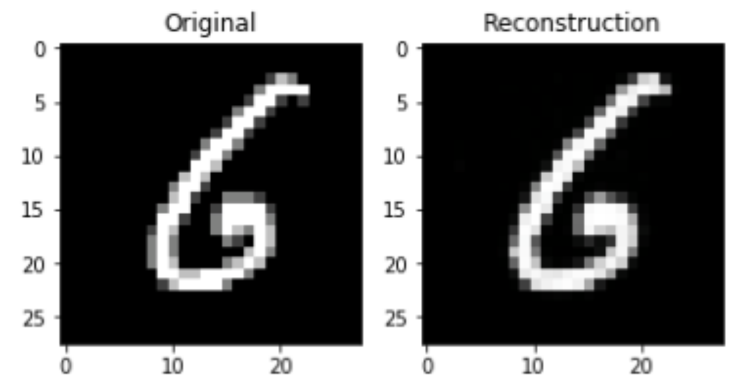


Fig. 3 Generated from a simple auto encoder, reconstructing the number “6”

The encoding and decoding functions learn the representations of the actual data. Below is the actual visualization of how the individual units learn the representations.

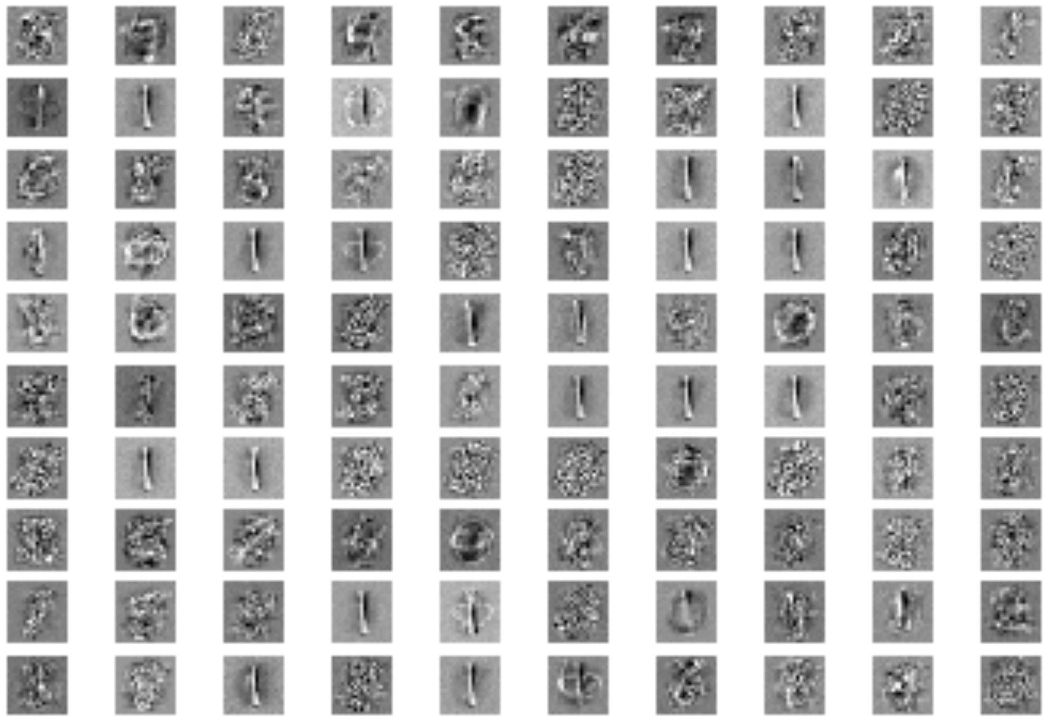


Fig. 4 Weights connecting the encoder and decoder

3.3 Di-entangling the style in latent space

The network has been made deeper. Additional hidden layers have been stacked on top of the hidden layers. The additional layers of units 400, 200, 100, 20, 10 respectively have been stacked on top of the input layer both at the encoder and the decoder. The latent space consisted of 10 Z(hidden vector)s. Each Z when stridden across a specific range of values, several styles have been revealed to to dis entangle. The below figure shows how number 6 showed different style across different Z.
When the sigmoid has been used as the activation function, the change in Zs(0 to 1) resulted into different digits. The change in the style/digit has been very rapid.

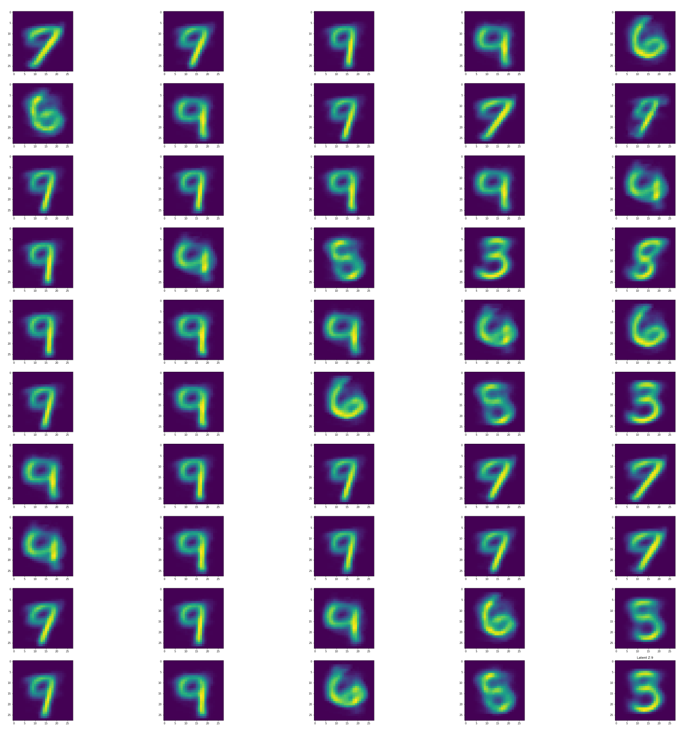


Fig. 5 Shows the output reconstructed images when values of Z have been changed across each dimension.

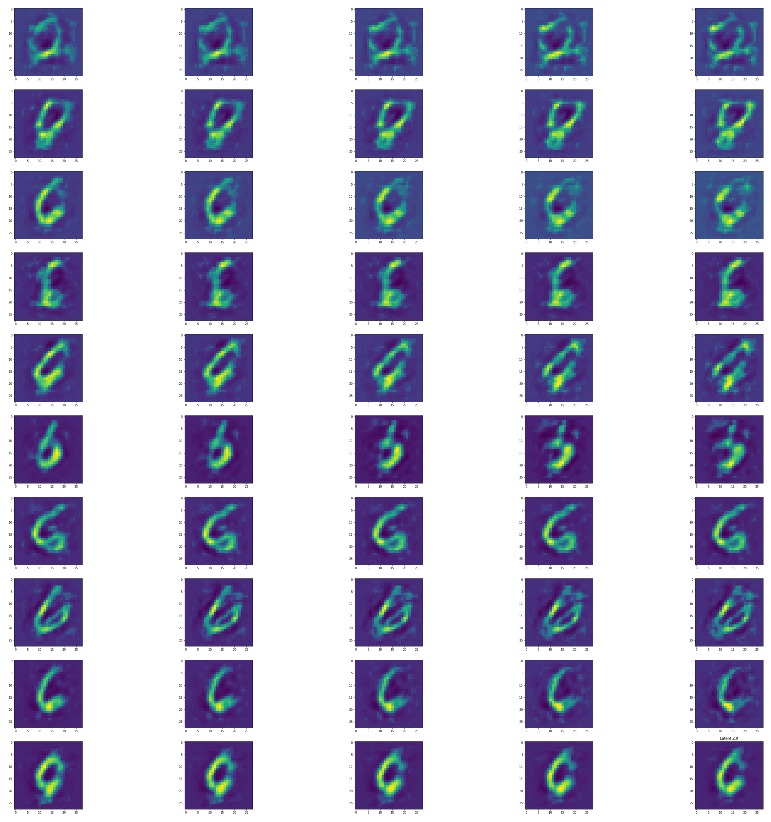


Fig. 6 Shows dis-entangling of styles of handwritten digit ‘6’ across various values of Z.

Leaky Relu function has been used for activation in the above picture, since deeper architectures are prone to gradient vanishing problems. The latent space is very vast within specific values. The below figure shows the concentration of each digit(0-9) in the latent space, when projected on to 3 dimensions.

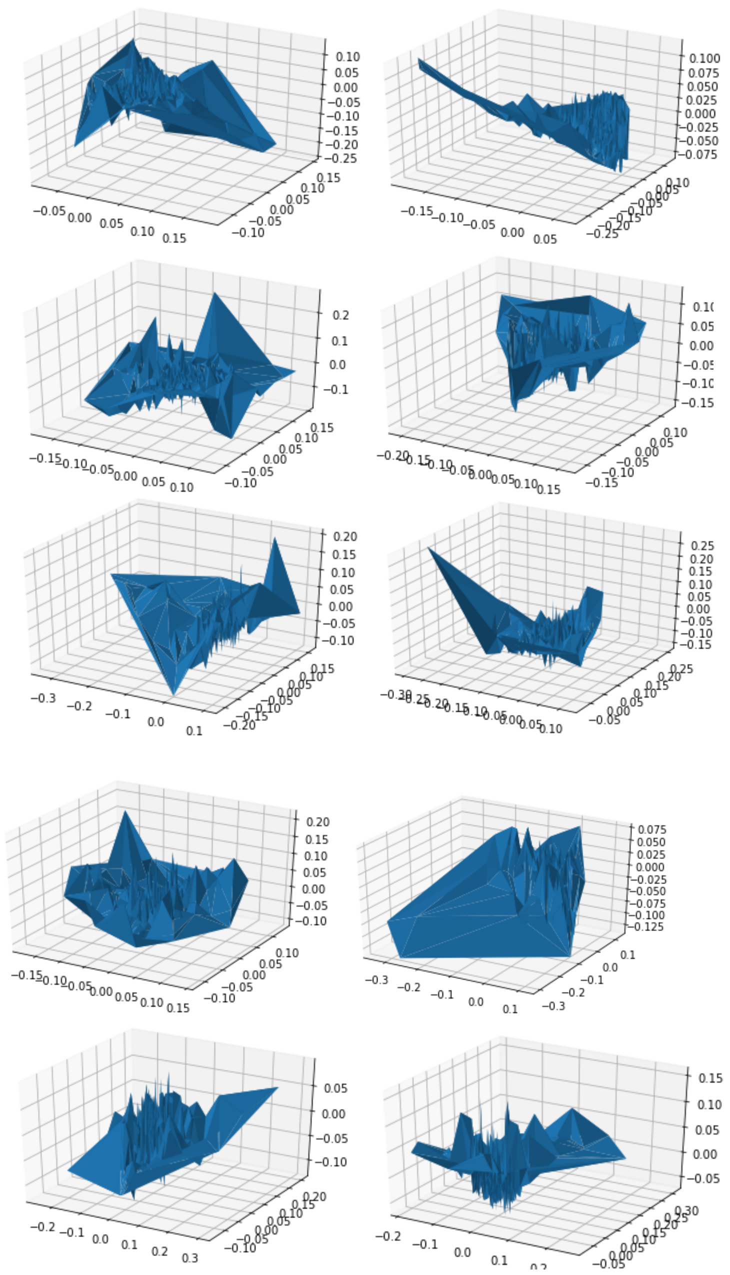


Fig. 7 3D Tri-Surf plots of data presence in latent space, starting from 0 to 9 respectively.

3.3 Sparse Auto- Encoder

Our interest in representation learning is to also enable the models to be generative. Simply copying the data through reconstruction will not provide us any advantage in generating the data. Sparse Auto- Encoders solve this by making the representations more general and less specific to the training data. The idea is to penalize the hidden unit biases. Penalizing the hidden unit values alone adds the risk of increased weights at these neurons to compensate this penalization. Thus weight matrices must also be penalized.

The sparse penalty ensures that less number of neurons are actually active at any time.

The learning process involves minimizing the loss function L(g(f(x)), x) +p(h), where p is the penalty on h(latent vectors). In our work, for the penalty term p(h), we apply KL (Kullback–Leibler) divergence with respect to a constant ρ.

KL divergence provides a measure for difference between 2 distributions. KL divergence between 2 distributions can be given by:

$KL(\rho 1 | \rho 2) = \int \rho 1 * \log(\rho 1/ \rho 2) + \int (1- \rho 1) * \log(1-\rho 1/ 1-\rho 2)$

In a typical model, the regularization represents adding prior to the model. However, with an auto encoder, we must see the sparsity penalization with a different view. The auto encoder is an approximation learning framework for data

$\mathrm{Log}(p_{model}(x)) = \Sigma\, p_{model}(h,x)$

From the encoder, we maximize the likelihood of encoding

$\mathrm{Log}\ p_{model}(h,x) = \mathrm{Log}\ p_{model}(h) + \mathrm{Log}\ p_{model}(x|h)$. From the below figure, we can see that the weights are very general. On average every unit learns more generalized representation from the data.

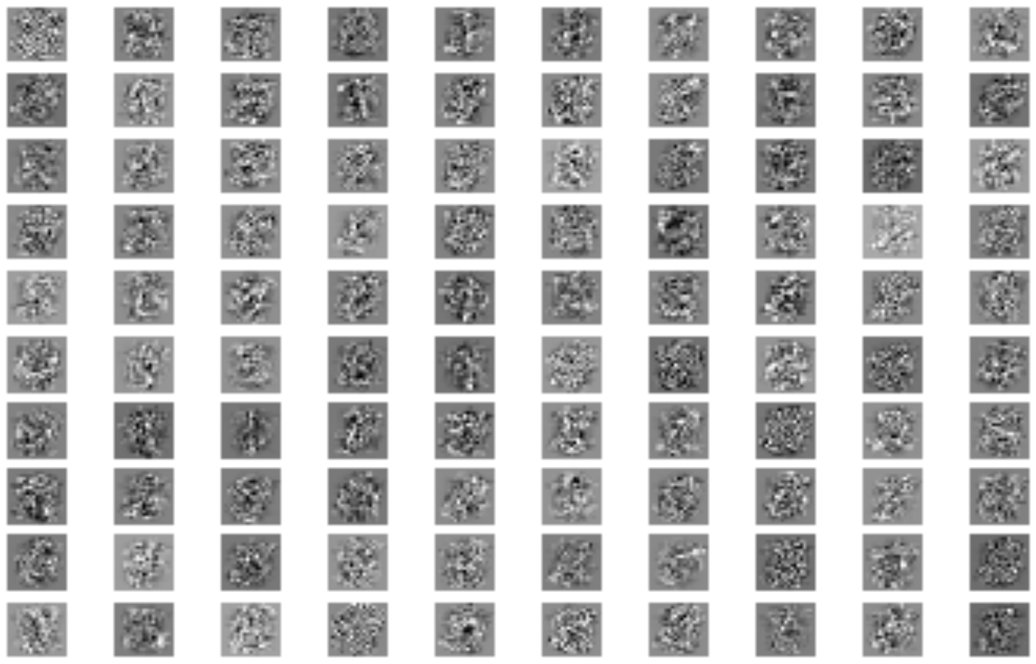


Fig. 8 Weights of sparse auto encoder.

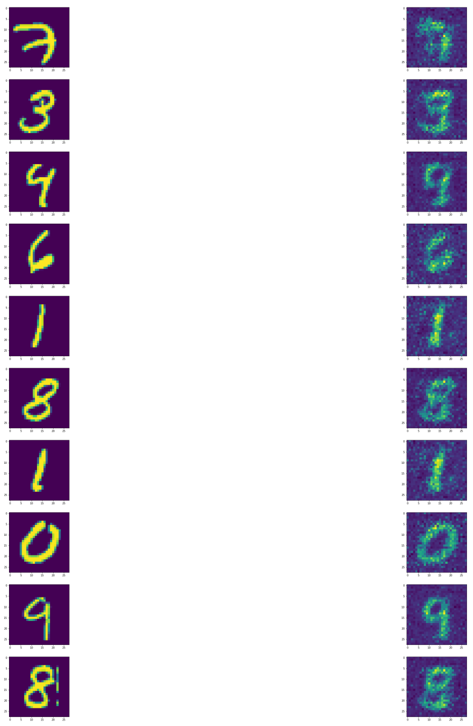


Fig. 9 Reconstruction of images after applying sparse encoding

With sparse encoding, the reconstruction losses have been high initially but after several epochs, the losses reach same in both the cases. This means even when constrained to learn with less number of activations, the Auto encoder is still good at reconstructing after several epochs. This improves the computational time for training deeper networks without much compromise in the error.

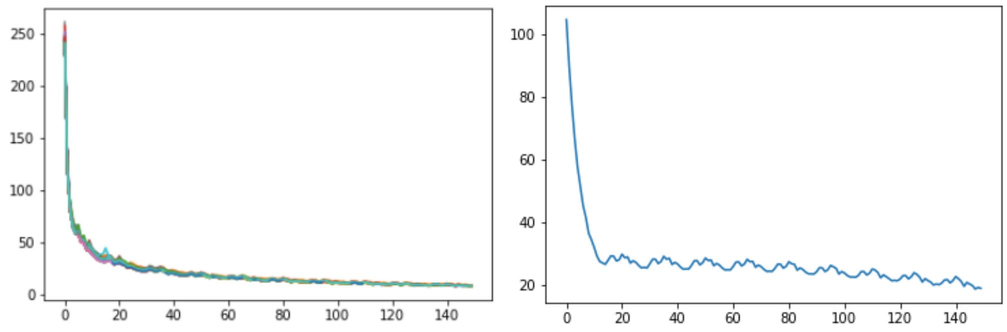


Fig. 10 Reconstruction losses with sparse encoding on left and with out sparse encoding on right.

3.4 Hyper parameter selection

The prior distribution we try to match is given by ρ

When ρ is 0, L_{kl} = -log(1-h),

ρ is 1, L_{kl}= -log(h).

Parameter: ρ	Reconstruction Loss
0.1	3211.51
0.2	2904.55
0.3	3179.76
0.4	3287.09
0.5	3073.50
0.6	2895.69
0.7	3115.06

Fig. 11 Shows a table with hyper parameter ρ and reconstruction loss over entire training.

Found the best reconstruction, when ρ is 0.2 and 0.6.

The beta term refers to the multiplier for the penalty added to the KL divergence term in the cost function.

Parameter: beta	Reconstruction Loss
1	3246.08
2	2900.48
3	3167.50
4	3062.56
5	3124.51
6	3112.30
7	3020.83

Fig. 12 Shows a table with hyper parameter ρ and reconstruction loss over entire training.

Found the best reconstruction, when beta is 2.

The alpha term refers to the multiplier used to the weight normalization (L2 norm) of the neural network.

Parameter: alpha	Reconstruction Loss
0.1	3025.95
0.2	3157.22
0.3	3008.36
0.4	3085.91
0.5	3248.07
0.6	3057.85
0.7	2943.99

Fig. 13 Shows a table with hyper parameter alpha and reconstruction loss over entire training.

Found the best reconstruction, when alpha is 0.3

3.5 Conclusion

Sparse encoder serves as a better generalization model for the auto encoder. It provides benefits in generalization, computation speed, with out much increase in the reconstruction loss. The dis-entangling of the features has been more explored with regular auto-encoder, sparse encoder requires fine tuning, and right set of hyper parameters have to be identified to train it better, more over with the deep architectures, typical penalty functions such as kl divergence must be updated, since deep architectures usually are more prone to gradient vanishing problems and sigmoid functions no longer serve, instead relus, leaky relus will be typically employed.

3.6 Variational Auto Encoder Design

Variational Auto Encoder(VAE) is a directed graphic model. The model consist of Encoder network and the decoder network. The actual function of a VAE is little different from a typical auto encoder. A VAE's encoder tries to learn the approximate inference, the training can be done by just gradient-based methods.

Essentially, if z is the latent vector for input x, encoder tries to approximate p(z) by q(z|x). This is achieved by the gradient-based training. From q(z|x), we sample latent vector z_i for each x_i during the training process. From these latent vectors z, the VAE's decoder tries to reconstruct the original input thereby generating a distribution p(x|z), from this distribution we again sample the final output.

The latent vectors are not unique vectors but a distribution, so they are sampled before passing them to the decoder as shown below.

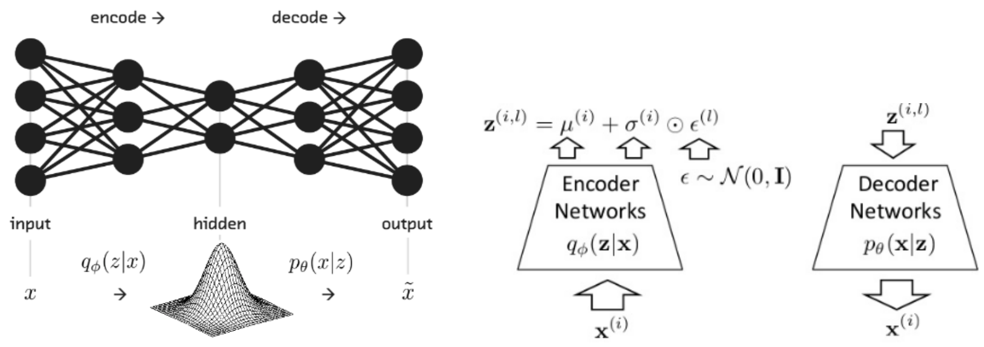

encode →
decode →
input
hidden
output
$q_\phi(z|x)$
$p_\theta(x|z)$
x
$\tilde{x}$
$\mathbf{z}^{(i,l)} = \mu^{(i)} + \sigma^{(i)} \odot \epsilon^{(l)}$
Encoder Networks $q_\phi(\mathbf{z}|\mathbf{x})$
$\epsilon \sim \mathcal{N}(0, \mathbf{I})$
$\mathbf{x}^{(i)}$
$\mathbf{z}^{(i,l)}$
Decoder Networks $p_\theta(\mathbf{x}|\mathbf{z})$
$\mathbf{x}^{(i)}$

Fig. 14 Shows basic architecture for the VAE, the left picture shows the entire architecture, the right picture shows the re-parameterization trick.

The training of the network is done by maximizing the variational lower bound L(q) associated with the data point x:

$$L(q) = E_{z\sim q(z|x)}(\log p_{model}(z, x)) + H(q(z \mid x))$$

$$= E_{z\sim q(z|x)}(\log p_{model}(x \mid z)) - D_{KL}(q(z \mid x)||p_{model}(z))$$

Here E is the expectation D_{KL} is the Kullback–Leibler divergence. This KL divergence term tries to represent the difference between the approximate posterior distribution q(z|x) and model prior.

The log probablility for the data is given by:

$$\mathrm{Log}(P(X))=KL(q(z|X)||p(z|X))+L\ (q)$$

By maximizing the lower bound we increase the probability of data, this in turn reduces the approximation error due to the learnt q(z|X). Thus our optimization task becomes maximizing this lower bound.

While approximating the distribution q(z|X) is one of it's true merits, VAE's also have few dis advantages. Samples drawn from VAEs are often blurry. This could be due to minimization of the difference between model distribution and data distribution, as a result data points not present in the training set also tend to receive higher probability mass function.

3.7 Implementation and Results

The entire architecture has been implemented in Tensorflow. The architecture used consists of sequence of hidden layers from 400, 100, 10, 2 in the encoder and all the way to the input size in the decoder. A separate class has been created for this autoencoder, this enables object orientented testing much easier. Now the user can instantiate any number of such VAE objects and all the tensorflow variables are contained with in the object, thus improving the debugging, testing capabilities. The structure of the VAE could be dynaically mentioned while calling the object.

After training the model the images are reconstructed as shown below

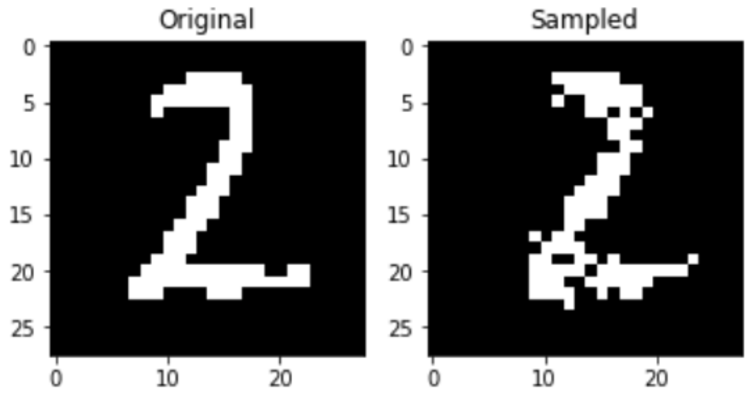


Fig. 15 Shows reconstruction of original image with digit 2 using the variational auto encoder.

The latent space's causal influence over the reconstruction can be visualized by the following picture, where the latent values are stridden across their range and their decoding through the decoder is plotted at each value of z1, z2(latent variables).

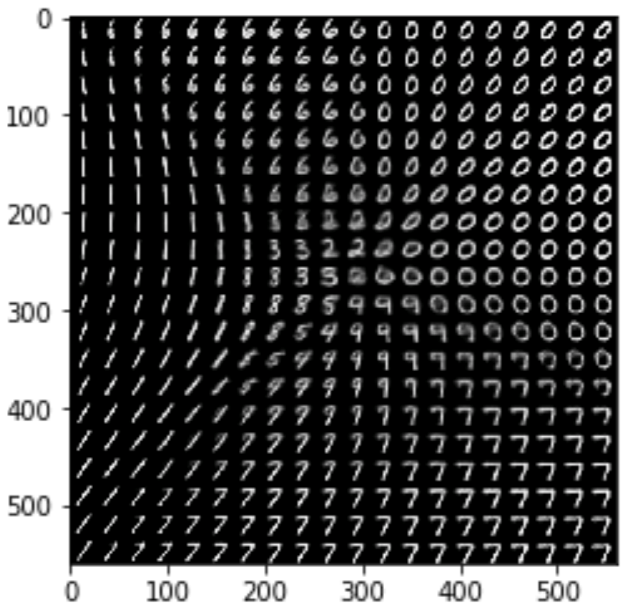


Fig. 16 Shows the projection of latent space along z(latent dimension) axes, after 20 epochs.

The below figure shows the losses occurred during the training. It is evident that the model is trained pretty quickly and the losses minimize within few epochs of the training.

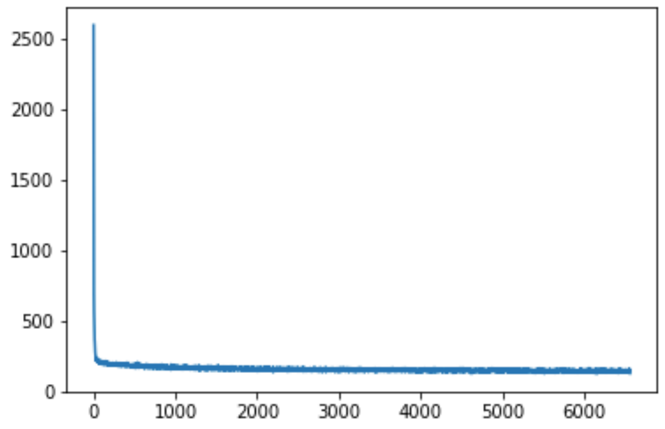


Fig. 17 Shows the losses along consecutive iterations during the training.

3.8 Conclusion

From the sampled pictures it has been observed that the samples have been little blurry. This could be due to assignment of high probability mass to data points which are not part of the training set/actual digit representation. This limitation could be better handled using adversarial networks, especially Adversarial Auto Encoders.

4.0 Generative adversarial network

Generative Adversarial networks provide alternative to maximum likelihood techniques, pixel wise independent means squared error techniques.

The generative adversarial network is based on a game theoretic scenario in which the generator network competes with the discriminator. The generator tries to produce samples, where as the discriminator tries to discriminate the samples if they are real samples or generated by the generator.

4.0.1 DC GAN

Deep Convolutional Generative Adversarial Networks are generative adversarial networks with certain architectural constraints (Convolutional Networks) which have shown evidence in the past to learn hierarchy of representations from object parts to scenes in datasets involving images.

The DC GAN has two units, a generator and a discriminator. The generator involves a de convolutional network which is supposed to convert random Z vector(100 dimensions) into a binary image of size 64x 64. The conversion of latent vector Z to actual image size is done by a fractionally strided de-convolution network. The role of a DC GAN is that the discriminator is supposed to identify the image from the generator as fake and the generator is supposed to make the discriminator believe that it is a true image. This adversarial training improves the generator over time and helps it learn important representations through its hidden layers.

The below figure shows the discriminator's structure.

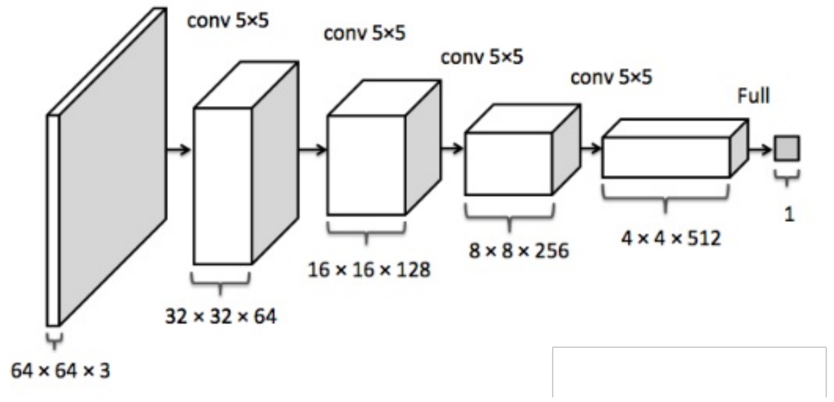


Fig. 18 Shows the discriminator architecture with series of convolution operations applied at each stage

4.1 Implementation and Results

The entire network architecture is very dynamic, we can mention the size or structure of the convolution block, de convolution block at run time while initializing the object.

4.1.1 Discriminator

The discriminator network includes several convolutional layers. All the convolution operations have been performed using tensor flow's built in libraries.

With every convolution operation, the output size of the tensor should be as follows:

$O = (W-K+2P)/S + 1$

Where W is the width of the input image, K is the kernel dimension, P is the padding size and S is the stride size. This formulation is due to the last convolution operation, the kernel might run out enough image pixels, therefore (W-(K-S))/S assuming zero padding.

The Tensor flow's convolution operations expects the tensor in 'NHWC' format, number of vectors present, height, width, color dimension respectively.

At the final stage all the convolutional layers are flattened and densely connected to an output unit to provide a binary result, whether the image is real or fake.

4.1.2 Generator

The generator uses a similar operation but instead performs de convolution. Where the feature map size increase with each de-convolution operation. This could also be termed as fractionally strided convolution operation, where the size of stride is a fraction (0.5).

Before arriving at the de-convolution operation, the random vector Z(latent vector), is densely connected with a network of size(f1xf1xd1), where f1 is the size of the feature map, d1 is the number of such feature maps/filters. From here the de convolution operation is applied and finally a fake image of original image's size is generated.

Below is the picture of images generated while training the network. It could be seen that after 400 iterations the image quality actually improved, this when the GAN became stable.

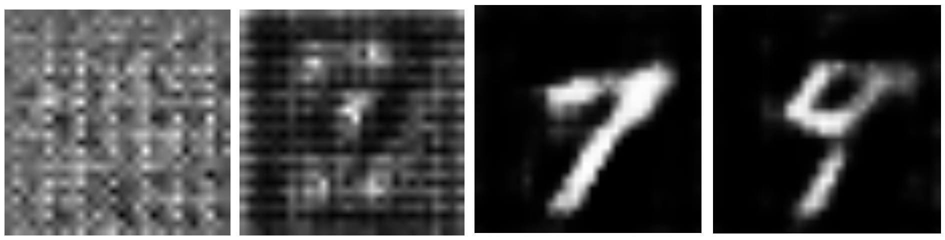


Fig. 19 Samples generated at 100, 200, 400 600 iterations respectively.

It is well known that the losses of discriminator or losses of generator are not true representatives of convergence/stability for the GANs. Both the generator cost and discriminator cost has to be lowered simultaneously. From the below figure, we can see that the GAN's both generator cost and discriminator cost converged after 1000 iterations. The generator cost has initially increased and later decreased, since it mastered how to generate realistic images. After 1000 iterations, both settled to a saddle point.

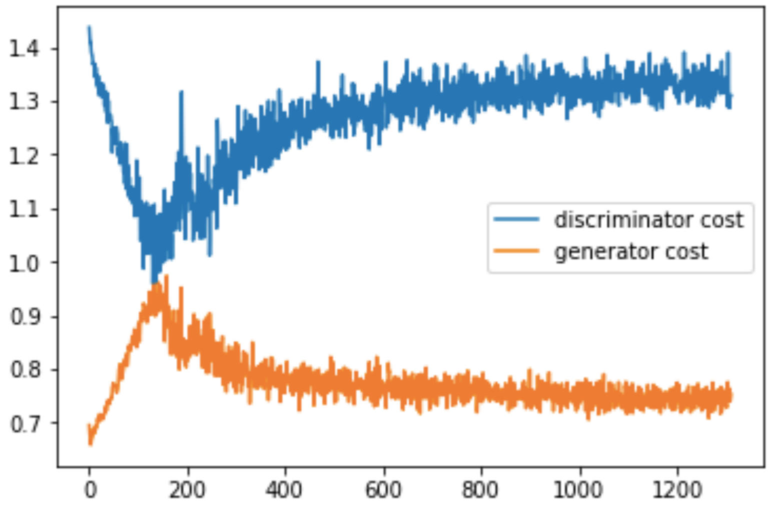


Fig. 20 shows discriminator and generator costs

4.2 Conclusion

The results show that DC-GANs are indeed good at generating good quality images. One remarkable capability of GAN training procedure is that, it doesn't just maximize the likelihood values of specific points(it may even assign negative or zero likelihood to certain points), instead it tries to learn the manifold of the training points. Usually the images generated by representing such manifolds are highly convincing to a human observer.

4.3 Adversarial Auto Encoder

Adversarial Auto encoder is a probabilistic auto encoder that uses generative adversarial network to perform the variational inference by trying to match the aggregated posterior of the hidden code vector of the auto encoder with an arbitrary prior distribution. This gives us a minimum guarantee that generating samples from any part of the prior space results into a meaningful image.

4.4 Adversarial Auto Encoder Design

The architecture of an AAE(Adversarial Auto Encoder) typically consists of a

generator and discriminator. The generator is a part of the auto encoder, that is the encoder part of the auto encoder acts as a generator here. The encoder used here consists of 2 hidden layers and the discriminator also consists of 2 hidden layers . The role of the discriminator here is to distinguish the fake latent vector, q(z) from the prior distribution p(z). while the discriminator's role is to generate hidden vectors which could resemble the random distribution p(z). While all this adversarial training is happening, it is the job of the decoder to reconstruct the input with minimal loss.

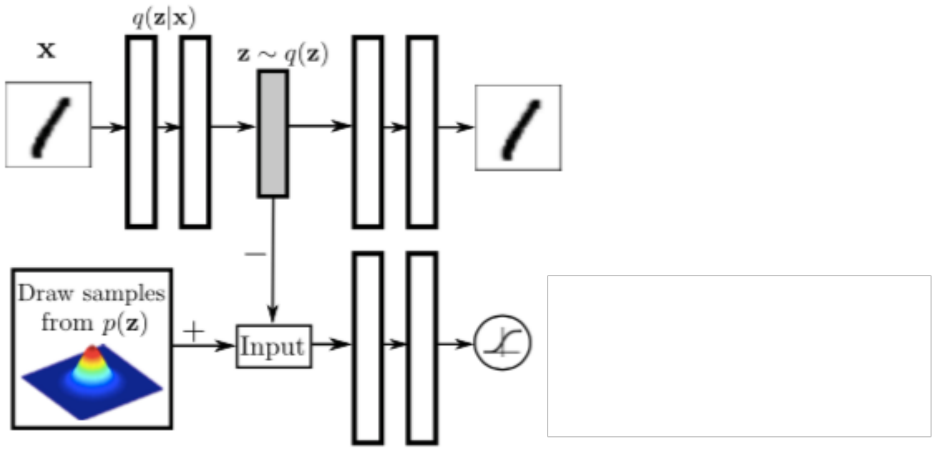

$\mathbf{x}$
$q(\mathbf{z}|\mathbf{x})$
$\mathbf{z} \sim q(\mathbf{z})$
$-$
Draw samples from $p(\mathbf{z})$
$+$
Input

Fig. 21 Shows the basic architecture of an adversarial auto encoder.

The role of encoder q(z|x), is to define an aggregated posterior distribution of

$q(z) = \int q(z|x)p_d(x)dx$, ($p_d$ represents the data distribution) the adversarial training ensures q(z) is similar to p(z), this could be considered as the regularization phase, since this is similar to sparse encoding we have seen previously. The decoding is considered as the reconstruction phase, both of these ensure that reasonable images could be generated from prior (z).

4.5 Implementation and results:

After performing 20 epochs the following results have been obtained. Both the generator and discriminator losses have been oscillating, after about 200 iterations they both settled to a stable value. Majority of the tools required to build the essential elements of AAE has been adopted from (https://github.com/hwalsuklee/tensorflow-mnist-AAE)

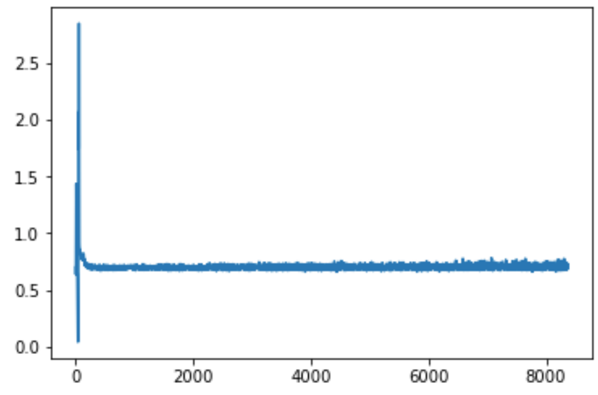


Fig. 22 Generator loss over the number of iterations

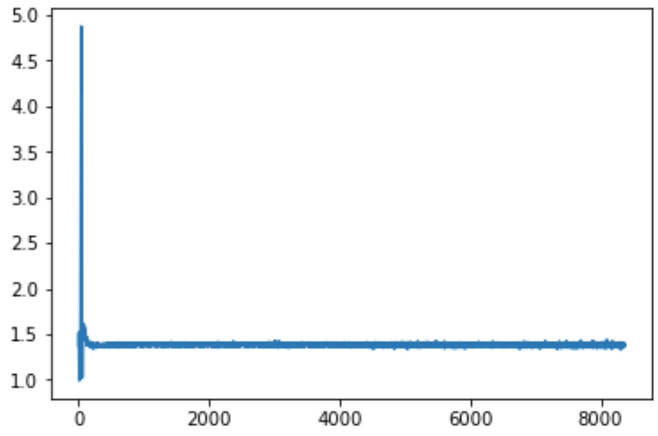


Fig. 23 Discriminator loss over the number of iterations

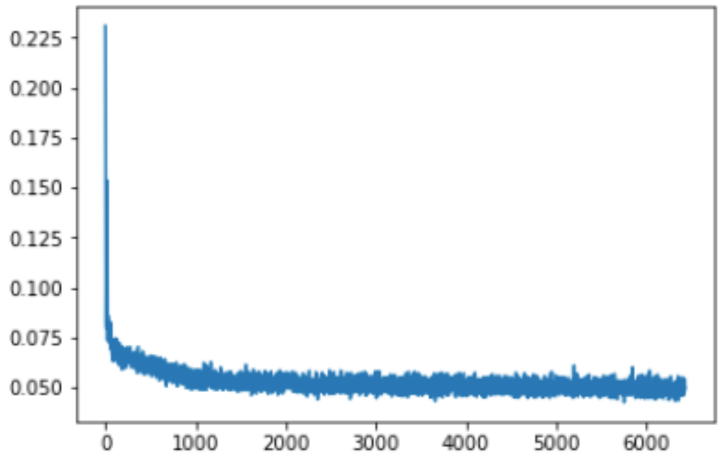


Fig. 24 Reconstruction loss over the number of iterations.

The effectiveness of unsupervised algorithms could be well measured by the accuracy of their performance over clustering of a data. The figures show the manifolds of Z vectors(dimension 2), when projected, clusters into 10 segments each representing a digit from 0-9.

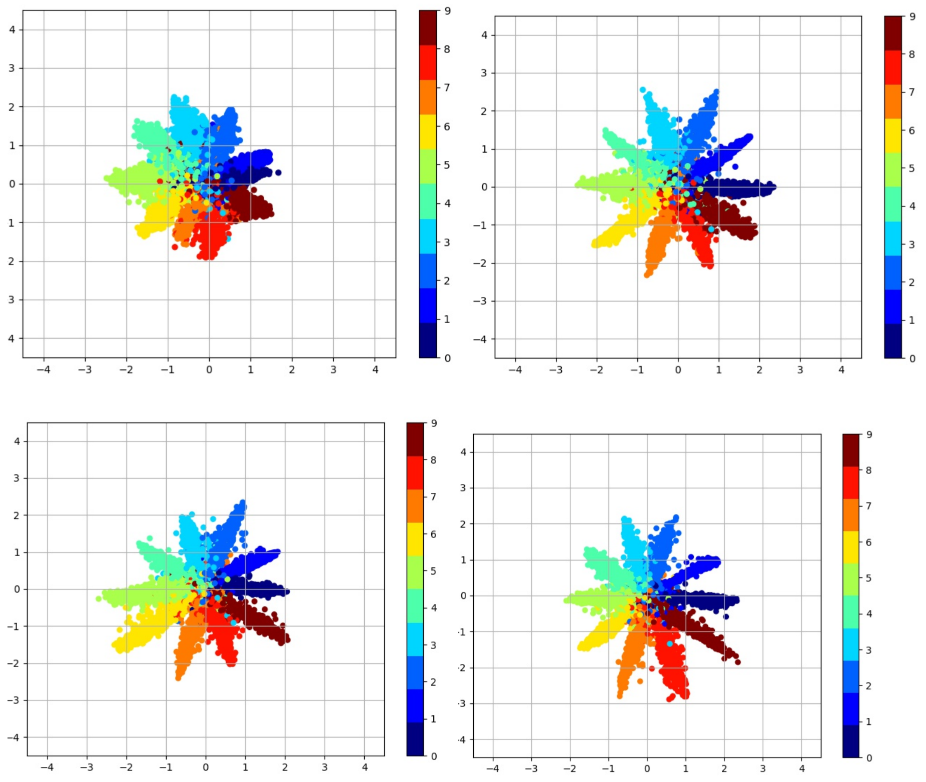


Fig. 25 Shows clustering across latent space with z =2 dimensions, from epoch 0, epoch 2, epoch 4 and epoch 6 respectively.

While the adversarial training has done the unsupervised learning well, the decoder has also shown good results in reconstructing the images from the latent space has been adversarially trained to be close the the prior distribution, p(z). Below are the results from reconstructing the images at various stages of the training.

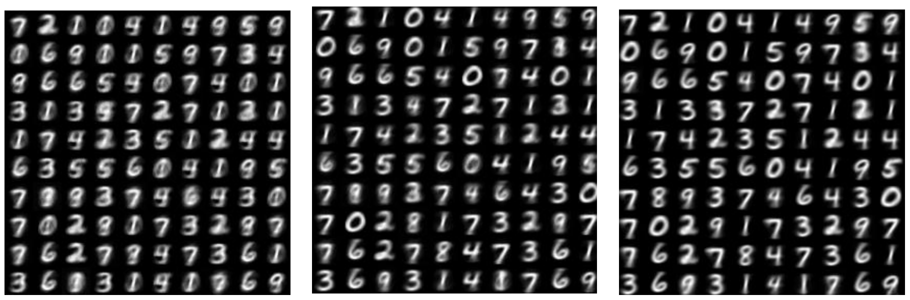


Fig. 26 Reconstruction of images at epoch 0, 2, 20 respectively.

4.6 Conclusion

It is interesting to find lot of commonness between the sparse auto encoders, variational auto encoders(VAEs) and adversarial auto encoders. The AAEs(Adversarial Auto Encoders) apply regularization to the hidden vectors, which is similar to KL divergence term in the VAEs. However it is important to realize that in VAEs and sparse encoders, we impose a prior distribution on the hidden vector, whereas in AAEs, we use adversarial training to achieve the same feet, there is no likelihood increasing process here. Also we can see that the AAEs capture the manifold of the data very well, better than VAEs.

5 Latent Space Representation Learning Transfer: Novel Approach

A new approach has been proposed to understand the implications of learning representations of images with different content on to same latent space. It has been already proved in that the past with MNIST dataset that different latent dimensions disentangle the style of the handwritten digits. But what about high level content in the information? In this paper we present results describing how high level content in images could be disentangled easily by pooling all images of different content into same latent space.

The below figure shows how 2 different sets of images with different content were pooled into the same latent space.

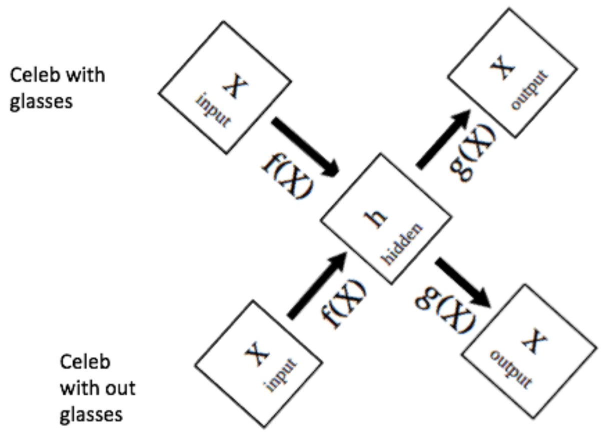


Fig. 27 Shows images with glasses and without glasses have been pooled into same latent space.

Essentially 2 auto encoders have been constructed which have same latent space but the training data differ by few high level content in the image.

5.1 Implementation and Results

The implementation of the auto-encoder has been purely in a object oriented style. The architecture of the auto encoder could be dynamically defined through the arguments passed in to it. This improves the code re usability, prototyping speed and debugging accuracy.

The cost function is still same, to minimize the reconstruction error.

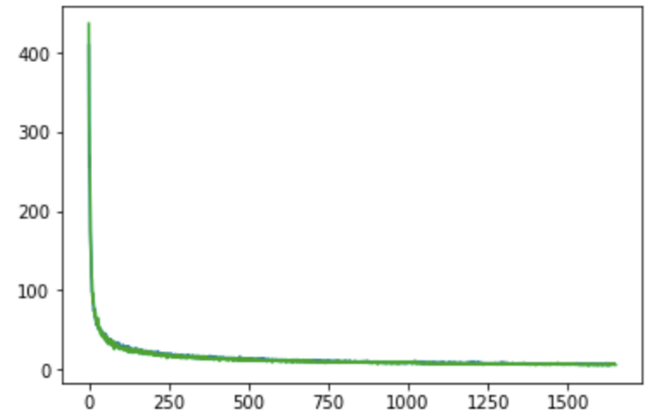


Fig. 28 Shows the reconstruction loss across the iterations.

After training the model for 10 epochs, test data has been sent through the encoder1(trained with data containing glasses) and the obtained latent vectors have been decoded through the decoder 2 (trained with data without glasses). The following results have been obtained.

Column 1 is the test image, column 2 is the obtained when decoded through it's own decoder. Column 3 is obtained when decoded through a different decoder which is completely unaware of the high level content, glasses.

When sparse coding, regularization is applied, the following results have been obtained.

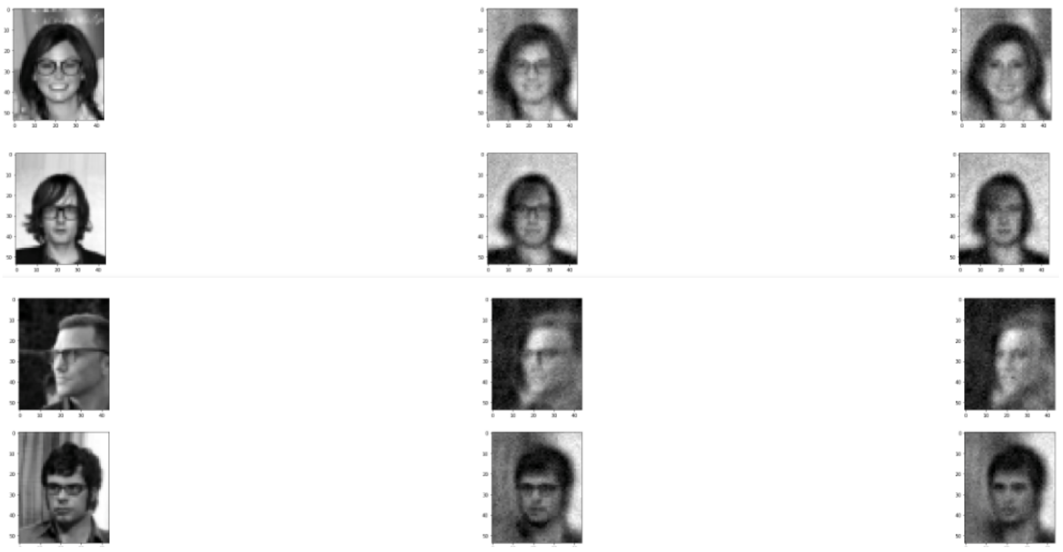


Fig. 29 Shows how the high level content from images has been removed after applying regularization and sparse encoding from a network with only 1 hidden layer

When the architecture is expanded with many deep hidden layers, the representations actually changed few facial features along with removing the glasses. This can seen from the below figure. This when the model tends to become more generative.

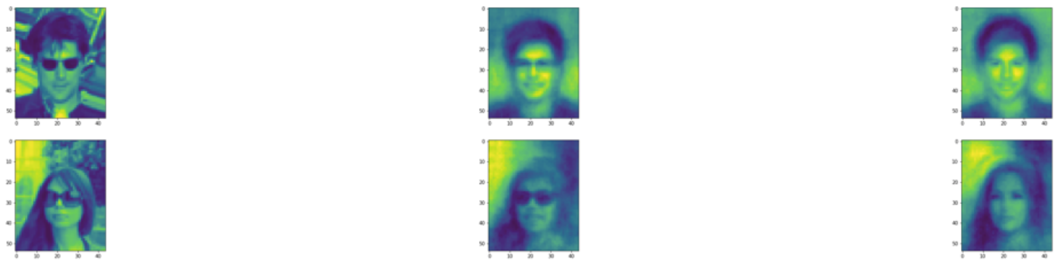


Fig. 30 Shows how the high level content from images has been removed along with few changes in facial features, from a network with deep architecture.

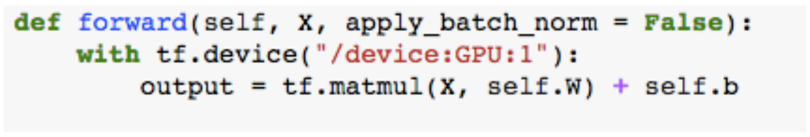
5.2 Conclusion

It's been interesting to see how using an auto encoder framework we could obtain/remove our desired high level content from the images. This increases our hope to have all our digital media communication with high level features customized to our interest, such as hair style, glasses, etc. where all the information transmitted would be the latent space and the endpoints perform the customer's preferred decoding.

6 GPU Parallelism

The computation of regular operations such as matrix multiplication, vector additions are often repetitive and can be computed most efficiently, if GPUs are utilized. Tensor flow libraries have efficient kernel implementations to achieve such parallelism. There are majorly two kinds of parallelism one could exploit using Tensor flow core kernels.

1. Model Parallelism: Here different GPUs run on different parts of the code/graph. Batches of data flow through the GPUs for each kind of computation. This kind of computation is best helpful when GPUs have different memory capacity, thereby enabling us to specify desired GPU for best performance.



```
def forward(self, X, apply_batch_norm = False):
    with tf.device("/device:GPU:1"):
        output = tf.matmul(X, self.W) + self.b
```

Fig. 32 Matrix multiplication, forced to run on GPU 1

2. Data Parallelism: Here we run same code on different GPUs with data distributed across the GPUs. This works best when all the GPUs hold equal memory capacity and there is no relative lag between each GPU.

After applying GPU settings(manual GPU placements) the performance of all the model trainings have improved by more than 5 %. Just for the CNN based Auto-Encoder training, the GPUs saved up to 5 minutes in time and when training multiple models, this scales up very quickly. The following picture shows how performance varies per each epoch when trained without GPU and with GPU.

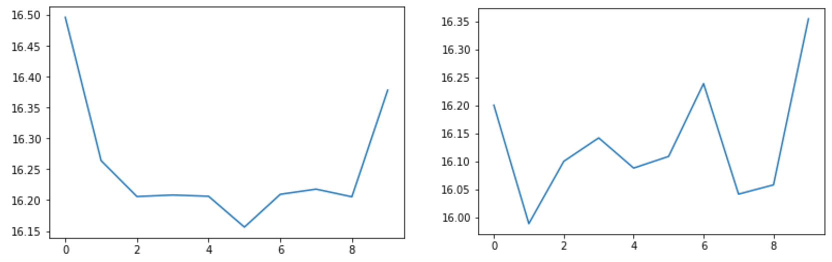


Fig. 32 Shows training time per each epoch without GPU placement, with GPU placement.

7 Final Analysis

After implementing various architectures and representation learning algorithms, it became very clear that all of them try to represent the data in a latent space, and all of them do this process by approximating the posterior distributions at the encoder/decoder. However adversarial networks stand out because they learn the manifolds better than VAEs and sparse auto encoders, especially the adversarial auto encoder seems to be very convincing in representing the best hidden vectors which provide better quality images while generating. Clearly adversarial auto encoders are winner here, however they might still have few downsides such as stability of the adversarial training and usually require fine tuning in the hyper parameters while training

Acknowledgments

Would like to thank our Prof. Sargur Srihari and his teaching assistants Mihir Chauhan, Mohammad Abuzar Shaikh, for designing such an interesting project and suggesting excellent ideas while struck at the roadblocks in the project.

References

[1] Yoshua Bengio, Aaron Courville, and Pascal Vincent Representation Learning: A Review and New Perspectives, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2014

[2] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. International Conference on Learning Representations (ICLR), 2014.

[3] Alireza Makhzani Jonathon Shlens Navdeep Jaitly Ian Goodfellow, Adversarial Autoencoders. International Conference on Learning Representations (2016).

https://www.tensorflow.org/

http://www.deeplearningbook.org/.