Adversarial Autoencoders

Using the proposed GAN's the authors of this paper have made an auto encoder in which performs variational inference by matching the aggregated posterior of the hidden code vector of the auto encoder with an arbitrary prior distribution. Using this technique, it enables the decoder to generate from any part of prior space results in meaningful samples. And this auto encoders have many applications such as classification or dimensionality reduction.

For generative models the task of capturing rich distributions is a critical task to perform. And until recently, many of the methods such as Restricted Boltzmann Machines (RBM), and Deep Belief Networks (DBNs) have been trained by MCMC-based algorithms. However, recently there have been discoveries of methods that enables training of generative model directly via back propagation. (such as VAE, importance weighted auto encoders, GAN or GMMM). In this paper the authors proposes a new method in which can turn an auto encoder into a generative model, called adversarial auto encoder. Mainly the auto encoders have two objectives, reconstruction loss as well as adversarial training loss that matches the aggregated posterior distribution of the latent representation of the auto encoder to an arbitrary prior distribution. As a result, the encoder learns how to encoder a given data into a prior distribution while the decoder learns a deep generative model that maps the imposed prior to the data distribution.

Generative Adversarial Networks

$$\min_G \max_D \mathrm{E}_{\mathrm{x}\sim p_{\mathrm{data}}}[\log D(\mathbf{x})] + \mathrm{E}_{\mathbf{z}\sim p(\mathbf{z})}[\log(1 - D(G(\mathbf{z}))]$$

In a very simple form there are two components, in a GAN, the generator network and discriminator network. The discriminator is a neural network that computes the probability that a point x in data space is a sample from the data distribution (positive samples) that we are trying to model, rather than a sample from our generative model (negative samples). And the generator is a network that maps samples z from the prior p(z) to the data space. And the main objective of the model can be expressed by the equation as seen above.

Adversarial Auto Encoders

$$q(\mathbf{z}) = \int_{\mathbf{x}} q(\mathbf{z}|\mathbf{x}) p_d(\mathbf{x}) d\mathbf{x}$$

q(z) be the prior distribution we want to impose on the

latent codes

q(z|x) be an encoding distribution

p(x|z) be an decoding distribution

pd(x) be the data distribution

p(x) be the model distribution

The encoding of an auto encoder function can be thought of as seen above, on the aggregated posterior distribution of **q(z)** on the hidden code vector. The AAE is an auto encoder that matches the aggregated posterior, q(z), to an arbitrary prior, p(z). And as seen below, to match the latent distribution of a given function, an adversarial portion can be added to a regular auto encoders. While the auto encoder tries to reconstruct the given data, the adversarial part tries to discriminate between the distribution given by the auto encoder or the distribution we wish to impose on the latent vector.

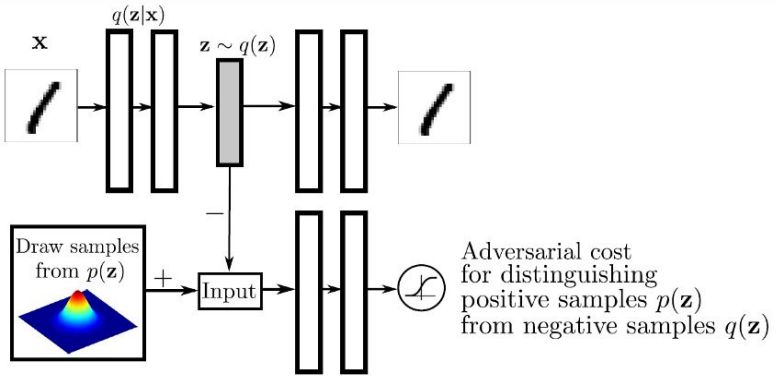


Figure 1: Architecture of an adversarial autoencoder. The top row is a standard autoencoder that reconstructs an image $\mathbf{x}$ from a latent code $\mathbf{z}$. The bottom row diagrams a second network trained to discriminatively predict whether a sample arises from the hidden code of the autoencoder or from a sampled distribution specified by the user.

Interesting note here is the fact that the generator portion of the auto encoder is the encoder portion of the auto encoder as well. As training continuous, the auto encoder will try to trick the discriminator and match the distribution of the given data. And in the training there are two phases, first the minimization of the reconstruction loss, and regularization phase where the network tries to match the distribution. Once the training procedure is done, the decoder of the auto encoder will define a generative model that maps the imposed prior of p(z). Choices of encoder portion of AAE includes...

1) **Deterministic**: Where the encoder is the same structure as the auto encoders, where the only stochasticity in q(z) is the data distribution, pd(x).

2) **Gaussian posterior**: This is similar VAE, where q(z|x) is a Gaussian distribution whose mean and variance is predicted by the encoder network. In this case, the stochasticity in q(z) comes from both the data-distribution and the randomness of the Gaussian distribution at the output of the encoder. (re-parametrization trick was used.)

3) **Universal approximator posterior**: we can use the auto encoders to train the encoder distribution of the network as the universal approximator of the posterior. If we set the encoder potion as f(x,n) where x is the input and n is a noise, with a fixed distribution, then we can assume q(z|x) = $\boldsymbol{\delta}$(z-f(x,n)), which we can define the encoder distribution to.

$$q(\mathbf{z}|\mathbf{x}) = \int_{\eta} q(\mathbf{z}|\mathbf{x},\eta)p_{\eta}(\eta)d\eta \quad \Rightarrow \quad q(\mathbf{z}) = \int_{\mathbf{x}}\int_{\eta} q(\mathbf{z}|\mathbf{x},\eta)p_d(\mathbf{x})p_{\eta}(\eta)d\eta d\mathbf{x}$$

And in the case above, we no longer have to assume that the distribution of the encoder is Gaussian (or fixed distribution we started off), and we can learn any arbitrary posterior distribution. (and this can be done via directly propagating the error.) Note that in the rest of the paper the authors only reports results from deterministic version.

Incorporating Label Information in the Adversarial Regularization

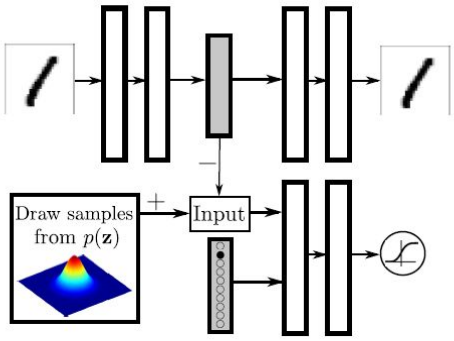


Figure 3: Regularizing the hidden code by providing a one-hot vector to the discriminative network. The one-hot vector has an extra label for training points with unknown classes.

In the case where we know the labels of the given data, that can aid us with forming the distribution of AAE. (As seen above.). A one hot encoded representation of the label is feed into the discriminator network, along side with the targeted distribution. One interesting thing to note here is the fact that in the one hot encoded vector, we reserve one additional space for the case where the given data does not have a label associated to it. During the positive phase the user can provide the labels of the mixture component to the discriminator through the one-hot vector. And during the negative phase the user can provide label of the

training point image to the discriminator through the one-hot vector.

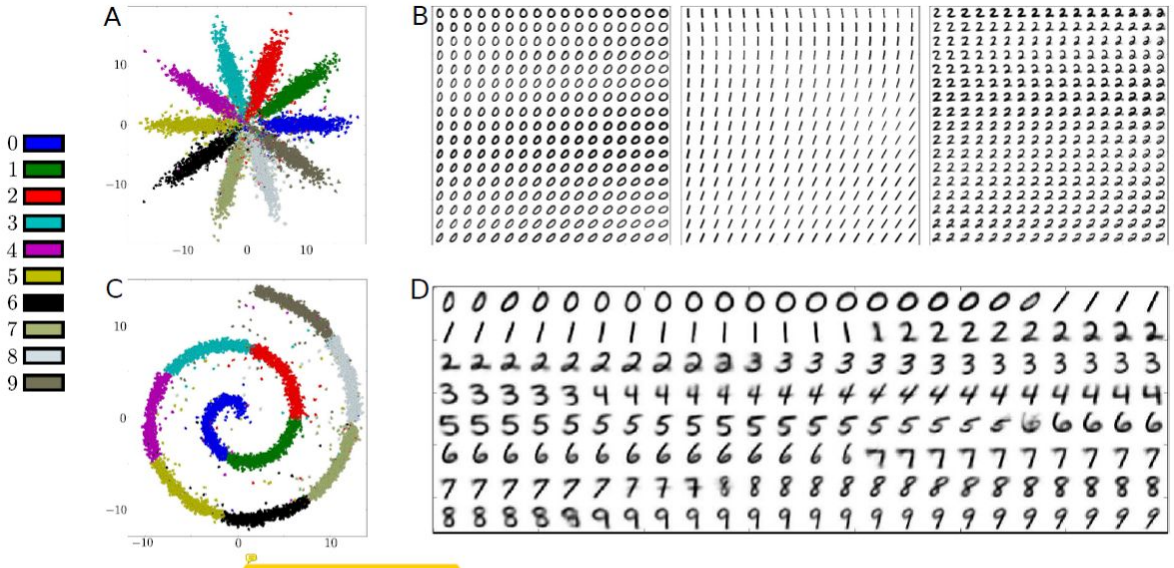


Figure 4: Leveraging label information to better regularize the hidden code. **Top Row:** Training the coding space to match a mixture of 10 2-D Gaussians: (a) Coding space **z** of the *hold-out* images. (b) The manifold of the first 3 mixture components: each panel includes images generated by uniformly sampling the Gaussian percentiles along the axes of the corresponding mixture component. **Bottom Row:** Same but for a swiss roll distribution (see text). Note that labels are mapped in a numeric order (i.e., the first 10% of swiss roll is assigned to digit 0 and so on): (c) Coding space **z** of the *hold-out* images. (d) Samples generated by walking along the main swiss roll axis.

Supervised Adversarial Autoencoders

Generative models, recently became one of the most popular approaches for semi-supervised learning since they can disentangle the class label information from many other latent factors of variation in a principled way. Here the authors describe how to separate the class label information from the image style information.

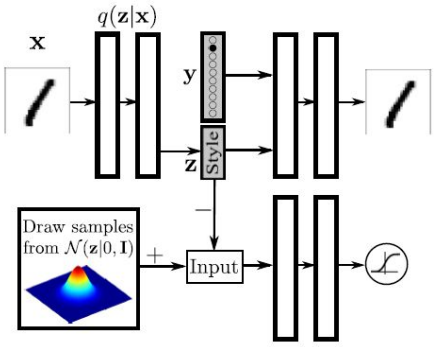
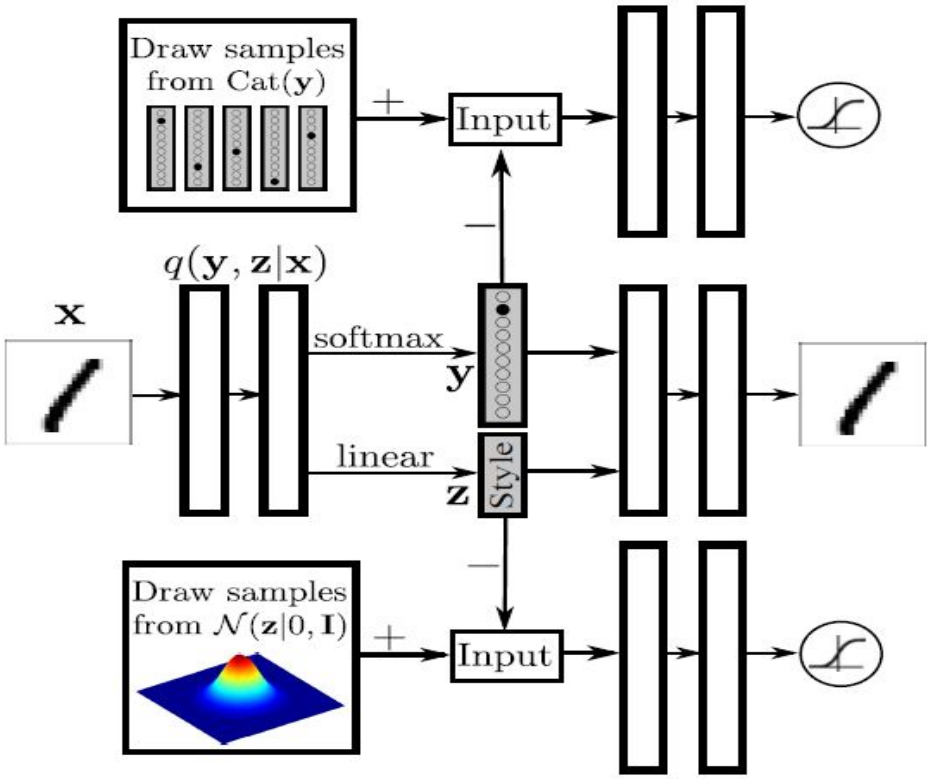


Figure 6: Disentangling the label information from the hidden code by providing the one-hot vector to the generative model. The hidden code in this case learns to represent the style of the image.

The way to achieve that is via something like above, where the class label is given to the generator network, so the encoder mainly have to learn the style.

Semi-Supervised Adversarial Autoencoders



Conclusion

In this paper, authors proposed to use the GAN framework as a variational inference algorithm for both discrete and continuous latent variables in probabilistic autoencoders. The method called the adversarial autoencoder (AAE), is a generative autoencoder that achieves competitive test likelihoods on real-valued MNIST and Toronto Face datasets. They discussed how this method can be extended to semi-supervised scenarios and showed that it achieves competitive semi-supervised classification performance on MNIST and SVHN datasets. Finally, they demonstrated the applications of adversarial autoencoders in disentangling the style and content of images, unsupervised clustering, dimensionality reduction and data visualization.