ADVERSARIAL AUTOENCODERS

Alireza Makhzani
University of Toronto
makhzani@psi.utoronto.ca

Jonathon Shlens & Navdeep Jaitly & Ian Goodfellow
Google Brain
{shlens,ndjaitly,goodfellow}@google.com

ABSTRACT

In this paper we propose a new method for regularizing autoencoders by imposing an arbitrary prior on the latent representation of the autoencoder. Our method, named "adversarial autoencoder", uses the recently proposed generative adversarial networks (GAN) in order to match the aggregated posterior of the hidden code vector of the autoencoder with an arbitrary prior. Matching the aggregated posterior to the prior ensures that there are no "holes" in the prior, and generating from any part of prior space results in meaningful samples. As a result, the decoder of the adversarial autoencoder learns a deep generative model that maps the imposed prior to the data distribution. We show how adversarial autoencoders can be used to disentangle style and content of images and achieve competitive generative performance on MNIST, Street View House Numbers and Toronto Face datasets.

1 INTRODUCTION

Building scalable generative models to capture rich distributions such as audio, images or video is one of the central challenges of machine learning. Until recently, deep generative models, such as Restricted Boltzmann Machines (RBM), Deep Belief Networks (DBNs) and Deep Boltzmann Machines (DBMs) were trained primarily by MCMC-based algorithms (Hinton et al., 2006; Salakhutdinov & Hinton, 2009). In these approaches the MCMC methods compute the gradient of log-likelihood which becomes more imprecise as training progresses. This is because samples from the Markov Chains are unable to mix between modes fast enough. In recent years, generative models have been developed that may be trained via direct back-propagation and avoid the difficulties that come with MCMC training. For example, variational autoencoders (VAE) (Kingma & Welling, 2014; Rezende et al., 2014) or importance weighted autoencoders (Burda et al., 2015) use a recognition network to predict the posterior distribution over the latent variables, generative adversarial networks (GAN) (Goodfellow et al., 2014) use an adversarial training procedure to directly shape the output distribution of the network via back-propagation and generative moment matching networks (GMMN) (Li et al., 2015) use a moment matching cost function to learn the data distribution.

In this paper we propose a general approach, called an adversarial autoencoder that can turn an autoencoder into a generative model. In our model, an autoencoder is trained with dual objectives – a traditional reconstruction error criterion, and an adversarial training criterion (Goodfellow et al., 2014) that matches the aggregated posterior distribution of the latent representation of the autoencoder to an arbitrary prior distribution. We show that this training criterion has a strong connection to VAE training. The result of the training is that the encoder learns to convert the data distribution to the prior distribution, while the decoder learns a deep generative model that maps the imposed prior to the data distribution.

1.1 GENERATIVE ADVERSARIAL NETWORKS

The Generative Adversarial Networks (GAN) (Goodfellow et al., 2014) framework establishes a min-max adversarial game between two neural networks – a generative model, G, and a discriminative model, D. The discriminator model, $D(\mathbf{x})$, is a neural network that computes the probability that a point $\mathbf{x}$ 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). Concurrently, the generator uses a function $G(\mathbf{z})$ that maps samples $\mathbf{z}$ from the prior $p(\mathbf{z})$ to the data space. $G(\mathbf{z})$ is trained

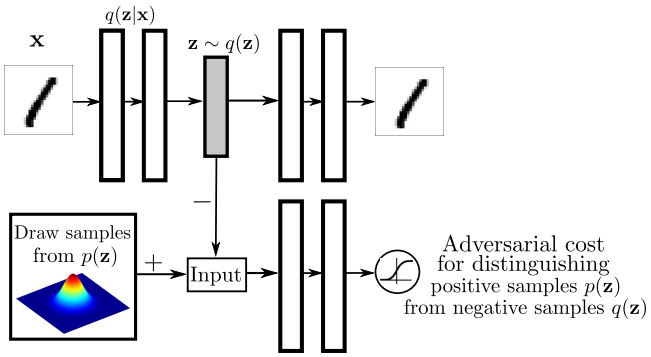


Figure 1: Architecture of an adversarial autoencoder. The top row is a standard autoencoder that reconstructs an image $\mathbf{x}$ from a latent code $\mathbf{z}$. The bottom row diagrams a second network trained to discriminatively predict whether a sample arises from the hidden code of the autoencoder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

to maximally confuse the discriminator into believing that samples it generates come from the data distribution. The generator is trained by leveraging the gradient of $D(\mathbf{x})$ w.r.t. $\mathbf{x}$, and using that to modify its parameters. The solution to this game can be expressed as following (Goodfellow et al., 2014):

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

The generator G and the discriminator D can be found using alternating SGD in two stages: (a) Train the discriminator to distinguish the true samples from the fake samples generated by the generator. (b) Train the generator so as to fool the discriminator with its generated samples.

2 Adversarial Autoencoders

Let $\mathbf{x}$ be the input and $\mathbf{z}$ be the latent code vector (hidden units) of an autoencoder with a deep encoder and decoder. Let $p(\mathbf{z})$ be the prior distribution we want to impose on the codes, $q(\mathbf{z}|\mathbf{x})$ be an encoding distribution and $p(\mathbf{x}|\mathbf{z})$ be the decoding distribution. Also let $p_d(\mathbf{x})$ be the data distribution, and $p(\mathbf{x})$ be the model distribution. The encoding function of the autoencoder $q(\mathbf{z}|\mathbf{x})$ defines an aggregated posterior distribution of $q(\mathbf{z})$ on the hidden code vector of the autoencoder as follows:

$$q(\mathbf{z}) = \int_{\mathbf{x}} q(\mathbf{z}|\mathbf{x}) p_d(\mathbf{x}) d\mathbf{x} \tag{1}$$

The adversarial autoencoder is an autoencoder that is regularized by matching the aggregated posterior, $q(\mathbf{z})$, to an arbitrary prior, $p(\mathbf{z})$. In order to do so, an adversarial network is attached on top of the hidden code vector of the autoencoder as illustrated in Figure 1. It is the adversarial network that guides $q(\mathbf{z})$ to match $p(\mathbf{z})$. The autoencoder, meanwhile, attempts to minimize the reconstruction error. The generator of the adversarial network is also the encoder of the autoencoder $q(\mathbf{z}|\mathbf{x})$. The encoder ensures the aggregated posterior distribution can fool the discriminative adversarial network into thinking that the hidden code $q(\mathbf{z})$ comes from the true prior distribution $p(\mathbf{z})$.

Both, the adversarial network and the autoencoder are trained jointly with SGD in two phases – the *reconstruction* phase and the *regularization* phase – executed on each mini-batch. In the reconstruction phase, the autoencoder updates the encoder and the decoder to minimize the reconstruction error of the inputs. In the regularization phase, the adversarial network first updates its discriminative network to tell the apart the true samples (generated using the prior) from the generated samples (the hidden codes computed by the autoencoder). The adversarial network then updates its generator (which is also the encoder of the autoencoder) to confuse the discriminative network.

Once the training procedure is done, the decoder of the autoencoder will define a generative model that maps the imposed prior of $p(\mathbf{z})$ to the the data distribution.

There are several possible choices for the encoder, $q(\mathbf{z}|\mathbf{x})$, of adversarial autoencoders:

Deterministic: Here we assume that $q(\mathbf{z}|\mathbf{x})$ is a deterministic function of $\mathbf{x}$. In this case, the encoder is similar to the encoder of a standard autoencoder and the only source of stochasticity in $q(\mathbf{z})$ is the data distribution, $p_d(\mathbf{x})$.

Gaussian posterior: Here we assume that $q(\mathbf{z}|\mathbf{x})$ is a Gaussian distribution whose mean and variance is predicted by the encoder network: $z_i \sim \mathcal{N}(\mu_i(\mathbf{x}), \sigma_i(\mathbf{x}))$. In this case, the stochasticity in $q(\mathbf{z})$ comes from both the data-distribution and the randomness of the Gaussian distribution at the output of the encoder. We can use the same re-parametrization trick of (Kingma & Welling, 2014) for back-propagation through the encoder network.

Universal approximator posterior: Adversarial autoencoders can be used to train the $q(\mathbf{z}|\mathbf{x})$ as the universal approximator of the posterior. Suppose the encoder network of the adversarial autoencoder is the function $f(\mathbf{x}, \eta)$ that takes the input $\mathbf{x}$ and a random noise η with a fixed distribution (e.g., Gaussian). We can sample from arbitrary posterior distribution $q(\mathbf{z}|\mathbf{x})$, by evaluating $f(\mathbf{x}, \eta)$ at different samples of η. In other words, we can assume $q(\mathbf{z}|\mathbf{x}, \eta) = \delta(\mathbf{z} - f(\mathbf{x}, \eta))$ and the posterior $q(\mathbf{z}|\mathbf{x})$ and the aggregated posterior $q(\mathbf{z})$ are defined as follows:

$$\begin{aligned} q(\mathbf{z}|\mathbf{x}) &= \int_{\eta} q(\mathbf{z}|\mathbf{x}, \eta) p_\eta(\eta) d\eta \\ q(\mathbf{z}) &= \int_{\mathbf{x}} \int_{\eta} q(\mathbf{z}|\mathbf{x}, \eta) p_d(\mathbf{x}) p_\eta(\eta) d\eta d\mathbf{x} \end{aligned} \tag{2}$$

In this case, the stochasticity in $q(\mathbf{z})$ comes from both the data-distribution and the random noise η at the input of the encoder. Note that in this case the posterior $q(\mathbf{z}|\mathbf{x})$ is no longer constrained to be Gaussian and the encoder can learn any arbitrary posterior distribution for a given input $\mathbf{x}$. Since there is an efficient method of sampling from the aggregated posterior $q(\mathbf{z})$, the adversarial training procedure can match $q(\mathbf{z})$ to $p(\mathbf{z})$ by direct back-propagation through the encoder network $f(\mathbf{x}, \eta)$.

Choosing different types of $q(\mathbf{z}|\mathbf{x})$ will result in different kinds of models with different training dynamics. For example, in the deterministic case of $q(\mathbf{z}|\mathbf{x})$, the network has to match $q(\mathbf{z})$ to $p(\mathbf{z})$ by only exploiting the stochasticity of the data distribution, but since the empirical distribution of the data is fixed by the training set, and the mapping is deterministic, this might produce a $q(\mathbf{z})$ that is not very smooth. However, in the Gaussian or universal approximator case, the network has access to additional sources of stochasticity that could help it in the adversarial regularization stage by smoothing out $q(\mathbf{z})$. Nevertheless, after extensive hyper-parameter search, we obtained similar test-likelihood with each type of $q(\mathbf{z}|\mathbf{x})$. So in the rest of the paper, we only report results with the deterministic version of $q(\mathbf{z}|\mathbf{x})$.

2.1 Relationship to Variational Autoencoders

Our work is similar in spirit to variational autoencoders (Kingma & Welling, 2014); however, while they use a KL divergence penalty to impose a prior distribution on the hidden code vector of the autoencoder, we use an adversarial training procedure to do so by matching the aggregated posterior of the hidden code vector with the prior distribution.

VAE (Kingma & Welling, 2014) minimizes the following upper-bound on the negative log-likelihood of $\mathbf{x}$:

$$\begin{aligned} E_{\mathbf{x}\sim p_d(\mathbf{x})}[-\log p(\mathbf{x})] &< E_{\mathbf{x}}[\mathrm{E}_{q(\mathbf{z}|\mathbf{x})}[-\log(p(\mathrm{x}|\mathrm{z})]] + E_{\mathbf{x}}[\mathrm{KL}(q(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))] \\ &= E_{\mathbf{x}}[\mathrm{E}_{q(\mathbf{z}|\mathbf{x})}[-\log p(\mathrm{x}|\mathrm{z})]] - E_{\mathbf{x}}[H(q(\mathbf{z}|\mathbf{x}))] + E_{q(\mathbf{z})}[-\log p(\mathbf{z})] \\ &= E_{\mathbf{x}}[\mathrm{E}_{q(\mathbf{z}|\mathbf{x})}[-\log p(\mathrm{x}|\mathrm{z})]] - E_{\mathbf{x}}[\sum_i \log\sigma_i(\mathbf{x}))] + E_{q(\mathbf{z})}[-\log p(\mathbf{z})] + \text{const.} \\ &= \text{Reconstruction} - \text{Entropy} + \text{CrossEntropy}(q(\mathbf{z}), p(\mathbf{z})) \end{aligned} \tag{3}$$

where the aggregated posterior $q(\mathbf{z})$ is defined in Eq. (1) and we have assumed $q(\mathbf{z}|\mathbf{x})$ is Gaussian and $p(\mathbf{z})$ is an arbitrary distribution. The variational bound contains three terms. The first term

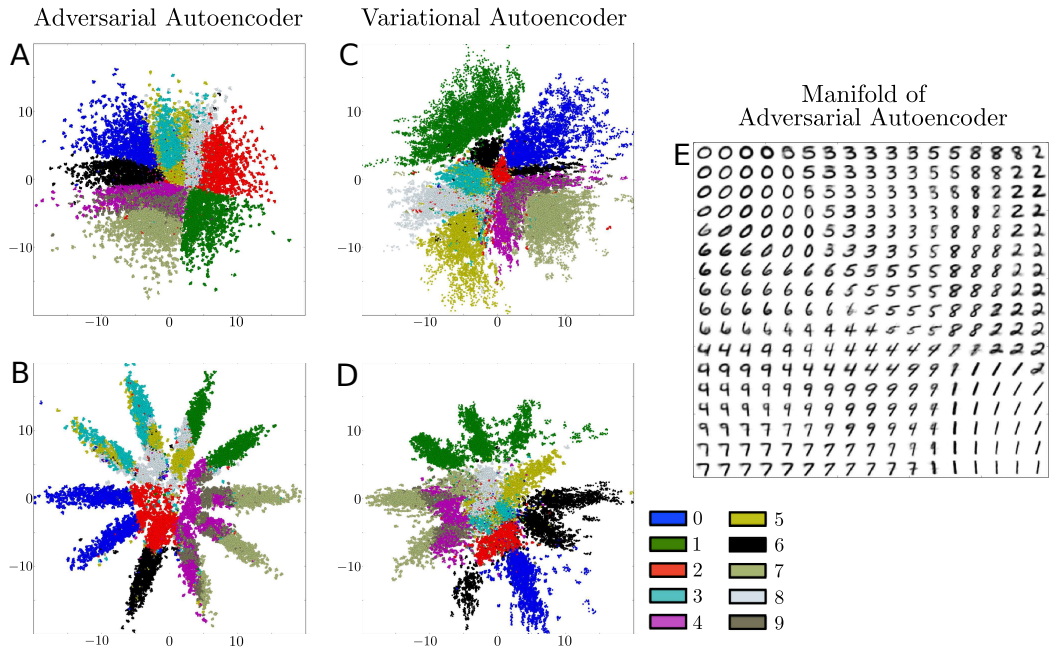


Figure 2: Comparison of adversarial and variational autoencoder on MNIST. The hidden code $\mathbf{z}$ of the *hold-out* images for an adversarial autoencoder fit to (a) a 2-D Gaussian and (b) a mixture of 10 2-D Gaussians. Each color represents the associated label. Same for variational autoencoder with (c) a 2-D gaussian and (d) a mixture of 10 2-D Gaussians. (e) Images generated by uniformly sampling the Gaussian percentiles along each hidden code dimension $\mathbf{z}$ in the 2-D Gaussian adversarial autoencoder.

can be viewed as the reconstruction term of an autoencoder and the second and third terms can be viewed as regularization terms. Without the regularization terms, the model is simply a standard autoencoder that reconstructs the input. However, in the presence of the regularization terms, the VAE learns a latent representation that is compatible with $p(\mathbf{z})$. The second term of the cost function encourages large variances for the posterior distribution while the third term minimizes the cross-entropy between the aggregated posterior $q(\mathbf{z})$ and the prior $p(\mathbf{z})$. KL divergence or the cross-entropy term in Eq. (3), encourages $q(\mathbf{z})$ to pick the modes of $p(\mathbf{z})$. In adversarial autoencoders, we replace the second two terms with an adversarial training procedure that encourages $q(\mathbf{z})$ to match to the whole distribution of $p(\mathbf{z})$.

In this section, we compare the ability of the adversarial autoencoder to the VAE to impose a specified prior distribution $p(\mathbf{z})$ on the coding distribution. Figure 2a shows the coding space $\mathbf{z}$ of the test data resulting from an adversarial autoencoder trained on MNIST digits in which a spherical 2-D Gaussian prior distribution is imposed on the hidden codes $\mathbf{z}$. The learned manifold in Figure 2a exhibits sharp transitions indicating that the coding space is filled and exhibits no "holes". In practice, sharp transitions in the coding space indicate that images generated by interpolating within $\mathbf{z}$ lie on the data manifold (Figure 2e). By contrast, Figure 2c shows the coding space of a VAE with the same architecture used in the adversarial autoencoder experiments. We can see that in this case the VAE roughly matches the shape of a 2-D Gaussian distribution. However, no data points map to several local regions of the coding space indicating that the VAE may not have captured the data manifold as well as the adversarial autoencoder.

Figures 2b and 2d show the code space of an adversarial autoencoder and of a VAE where the imposed distribution is a mixture of 10 2-D Gaussians. The adversarial autoencoder successfully matched the aggregated posterior with the prior distribution (Figure 2b). In contrast, the VAE exhibit systematic differences from the mixture 10 Gaussians indicating that the VAE emphasizes matching the modes of the distribution as discussed above (Figure 2d).

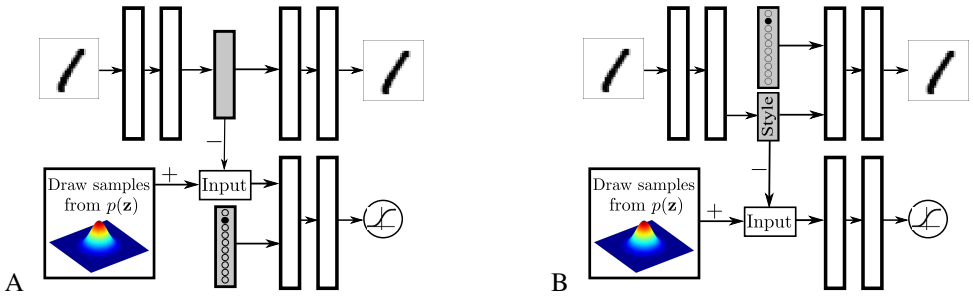


Figure 3: Two methods for semi-supervised learning with an adversarial autoencoder (a) Regularizing the hidden code by providing a one-hot vector to the discriminative network. The one-hot vector has an extra label for training points with unknown classes. (b) Disentangling the label information from 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 (see text).

An important difference between VAEs and adversarial autoencoders is that in VAEs, in order to back-propagate through the KL divergence by Monte-Carlo sampling, we need to have access to the exact functional form of the prior distribution. However, in adversarial autoencoders, we only need to be able to sample from the prior distribution in order to induce $q(\mathbf{z})$ to match $p(\mathbf{z})$. In Section 3, we demonstrate that the adversarial autoencoder can impose complicated distributions without having access to the explicit functional form of the distribution.

2.2 Relationship to GANs and GMMNs

In the original generative adversarial networks (GAN) paper (Goodfellow et al., 2014), GANs were used to impose the data distribution at the pixel level on the output layer of a neural network. Adversarial autoencoders, however, rely on the autoencoder training to capture the data distribution. In adversarial training procedure of our method, a much simpler distribution (e.g., Gaussian as opposed to the data distribution) is imposed in a much lower dimensional space (e.g., 20 as opposed to 1000) which results in a better test-likelihood as is discussed in Section 4.

Generative moment matching networks (GMMN) (Li et al., 2015) use the maximum mean discrepancy (MMD) objective to shape the distribution of the output layer of a neural network. The MMD objective can be interpreted as minimizing the distance between all moments of the model distribution and the data distribution. It has been shown that GMMNs can be combined with pre-trained dropout autoencoders to achieve better likelihood results (GMMN+AE). Our adversarial autoencoder also relies on the autoencoder to capture the data distribution. However, the main difference of our work with GMMN+AE is that the adversarial training procedure of our method acts as a regularizer that shapes the code distribution while training the autoencoder from scratch; whereas, the GMMN+AE model first trains a standard dropout autoencoder and then fits a distribution in the code space of the pre-trained network. In Section 4, we will show that the test-likelihood achieved by the joint training scheme of adversarial autoencoders outperforms the test-likelihood of GMMN and GMMN+AE on MNIST and Toronto Face datasets.

3 Semi-Supervised Adversarial Autoencoders

In scenarios where data is completely or partially labeled, adversarial training can incorporate the label to better shape the distribution of the hidden codes $\mathbf{z}$. We show two different ways of incorporating the label information into adversarial autoencoders. The labels can be incorporated either in the adversarial training process (regularization phase), or as additional latent variables in the reconstruction phase.

3.1 Incorporating Label Information in the Adversarial Training

We first describe how to leverage partial or complete label information to regularize the latent representation of the autoencoder more heavily. To demonstrate this architecture we return to Figure 2b

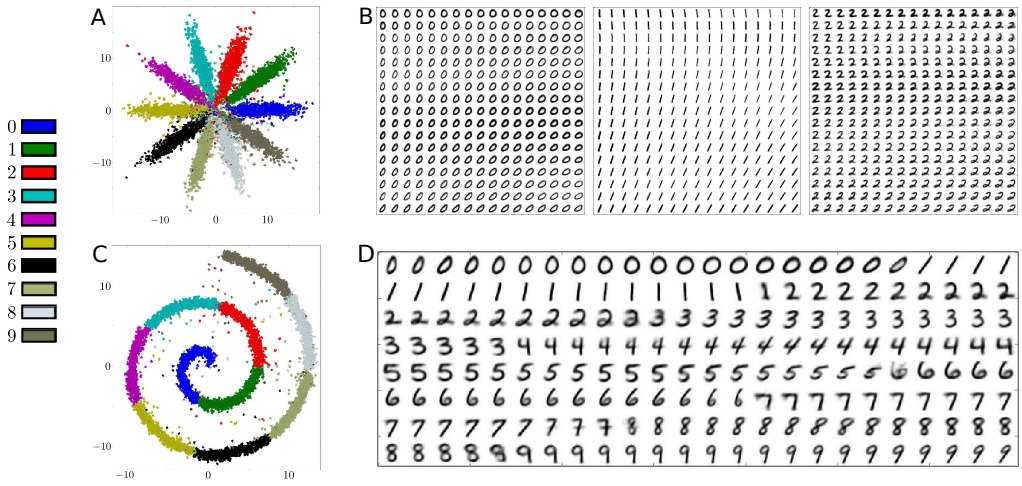


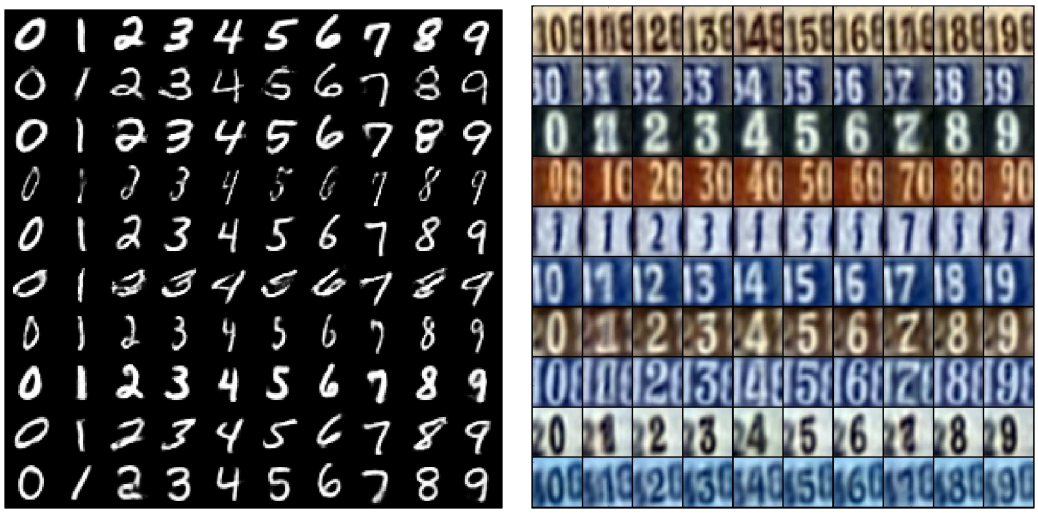
Figure 4: Leveraging label information to better regularize the hidden code. **Top Row:** Training the coding space to match a mixture of 10 2-D Gaussians: (a) Coding space $\mathbf{z}$ of the *hold-out* images. (b) The manifold of the first 3 mixture components: each panel includes images generated 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 $\mathbf{z}$ of the *hold-out* images. (d) Samples generated by walking along the main swiss roll axis.

in which the adversarial autoencoder is fit to a mixture of 10 2-D Gaussians. We now aim to force each mode of the mixture of Gaussian distribution to represent a single label of MNIST.

Figure 3a demonstrates the training procedure for this semi-supervised approach. We add a one-hot vector to the input of the discriminative network to associate the label with a mode of the distribution. The one-hot vector acts as switch that selects the corresponding decision boundary of the discriminative network given the class label. This one-hot vector has an extra class for unlabeled examples. For example, in the case of imposing a mixture of 10 2-D Gaussians (Figure 2b and 4a), the one hot vector contains 11 classes. Each of the first 10 class selects a decision boundary for the corresponding individual mixture component. The extra class in the one-hot vector corresponds to unlabeled training points. When an unlabeled point is presented to the model, the extra class is turned on, to select the decision boundary for the full mixture of Gaussian distribution. During the positive phase of adversarial training, we provide the label of the mixture component (that the positive sample is drawn from) to the discriminator through the one-hot vector. The positive samples fed for unlabeled examples come from the full mixture of Gaussian, rather than from a particular class. During the negative phase, we provide the label of the training point image to the discriminator through the one-hot vector.

Figure 4a shows the latent representation of an adversarial autoencoder trained with a prior that is a mixture of 10 2-D Gaussians trained on 10K labeled MNIST examples and 40K unlabeled MNIST examples. In this case, the i-th mixture component of the prior has been assigned to the i-th class in a semi-supervised fashion. Figure 4b shows the manifold of the first three mixture components. Note that the style representation is consistently represented within each mixture component, independent of its class. For example, the upper-left region of all panels in Figure 4b correspond to the upright writing style and lower-right region of these panels correspond to the tilted writing style of digits.

This method may be extended to arbitrary distributions with no parametric forms – as demonstrated by mapping the MNIST data set onto a "swiss roll" (Roweis & Saul, 2000) (a conditional Gaussian distribution whose mean is uniformly distributed along the length of a swiss roll axis). Figure 4c depicts the coding space $\mathbf{z}$ and Figure 4d highlights the images generated by walking along the swiss roll axis in the latent space.



(a) MNIST (b) SVHN

Figure 5: Disentangling content and style (15-D Gaussian) on MNIST and SVHN datasets.

3.2 Incorporating Label Information as Additional Latent Variables

Many latent factors of variation interact to generate an image. Recently, semi-supervised variational autoencoders have shown that the style and content of the images can be disentangled using an additional supervised cost (Kingma et al., 2014; Cheung et al., 2014). We now show that adversarial autoencoders may also be used to separate the image style from the class label information.

We alter the network architecture to provide a one-hot vector encoding of the label to the decoder (Figure 3b). The decoder utilizes both the one-hot vector identifying the label and the hidden code $\mathbf{z}$ to reconstruct the image. This architecture forces the network to retain all information independent of the label in the hidden code $\mathbf{z}$.

Figure 5a demonstrates the results of such a network trained on MNIST digits in which the hidden code is forced into a 15-D Gaussian. Each row of Figure 5a presents reconstructed images in which the hidden code $\mathbf{z}$ is fixed to a particular value but the label is systematically explored. Note that the style of the reconstructed images are consistent across a given row. Figure 5b demonstrates the same experiment applied to Street View House Numbers dataset (Netzer et al., 2011). A video showing the learnt SVHN style manifold can be found at `http://www.comm.utoronto.ca/~makhzani/adv_ae/svhn.gif`. In this experiment, the one-hot vector represents the label associated with the *central* digit in the image. Note that the style information in each row contains information about the labels of the left-most and right-most digits because the left-most and right-most digits are not provided as label information in the one-hot encoding.

4 Likelihood Analysis

The experiments presented in the previous sections have only demonstrate qualitative results. Several methods exist for quantitatively analyzing the quality of a generative model (Buades et al., 2005; Lyu & Simoncelli, 2009). In this section we measure the ability of the generative model to capture the data distribution by comparing the likelihood of this model to generate hold-out images on the MNIST and Toronto face dataset (TFD) using the evaluation procedure described in (Goodfellow et al., 2014).

We trained an adversarial autoencoder on MNIST and TFD in which the model imposed a high-dimensional Gaussian distribution on the underlying hidden code. Figure 6 shows samples drawn from the adversarial autoencoder trained on these datasets. A video showing the learnt TFD manifold can be found at `http://www.comm.utoronto.ca/~makhzani/adv_ae/tfd.gif`. To

(a) MNIST samples (8-D Gaussian)

(b) TFD samples (15-D Gaussian)

Figure 6: Samples generated from an adversarial autoencoder trained on MNIST and Toronto Face dataset (TFD). The last column shows the closest training images in pixel-wise Euclidean distance to those in the second-to-last column.

determine whether the model is over-fitting by copying the training data points, we used the last column of these figures to show the nearest neighbors, in Euclidean distance, to the generative model samples in the second-to-last column.

We evaluate the performance of the adversarial autoencoder by computing its log-likelihood on the hold out test set [1]. Evaluation of the model using likelihood is not straightforward because we can not directly compute the probability of an image. Thus, we calculate a lower bound of the true log-likelihood using the methods described in prior work (Bengio et al., 2013; 2014; Goodfellow et al., 2014). We fit a Gaussian Parzen window (kernel density estimator) to $10,000$ samples generated from the model and compute the likelihood of the test data under this distribution. The free-parameter σ of the Parzen window is selected via cross-validation.

Table 1 compares the log-likelihood of the adversarial autoencoder for real-valued MNIST and TFD to many state-of-the-art methods including DBN (Hinton et al., 2006), Stacked CAE (Bengio et al., 2013), Deep GSN (Bengio et al., 2014), Generative Adversarial Networks (Goodfellow et al., 2014) and GMMN + AE (Li et al., 2015).

	MNIST (10K)	MNIST (10M)	TFD (10K)	TFD (10M)
DBN (Hinton et al., 2006)	138 ± 2	-	1909 ± 66	-
Stacked CAE (Bengio et al., 2013)	121 ± 1.6	-	2110 ± 50	-
Deep GSN (Bengio et al., 2014)	214 ± 1.1	-	1890 ± 29	-
GAN (Goodfellow et al., 2014)	225 ± 2	386	2057 ± 26	-
GMMN + AE (Li et al., 2015)	282 ± 2	-	2204 ± 20	-
Adversarial Autoencoder	$\mathbf{340 \pm 2}$	**427**	$\mathbf{2252 \pm 16}$	**2522**

Table 1: Log-likelihood of test data on MNIST and Toronto Face dataset. Higher values are better. On both datasets we report the Parzen window estimate of the log-likelihood obtained by drawing 10K or 10M samples from the trained model. For MNIST, we compare against other models on the real-valued version of the dataset.

[1] Training details: the encoder, decoder and discriminator each have two layers of 1000 hidden units with ReLU activation function. The autoencoder is trained with a Euclidean cost function for reconstruction. The dimensionality of the hidden code $\mathbf{z}$ is 8 and 15 and the standard deviation of the Gaussian prior to be 5 and 10 for MNIST and TFD, respectively. On the Toronto Face dataset, data points are subtracted by the mean and divided by the standard deviation along each input dimension across the whole training set to normalize the contrast. However, after obtaining the samples, we rescaled the images (by inverting the pre-processing stage) to have pixel intensities between 0 and 1 so that we can have a fair likelihood comparison with other methods.

Note that the Parzen window estimate is a lower bound on the true log-likelihood and the tightness of this bound depends on the number of samples drawn. To obtain a comparison with a tighter lower bound, we additionally report Parzen window estimates evaluated with 10 million samples for both the adversarial autoencoders and the generative adversarial network (Goodfellow et al., 2014). In all comparisons we find that the adversarial autoencoder achieves superior log-likelihoods to competing methods.

5 CONCLUSION

In this paper we proposed a general framework to turn any autoencoder into a generative model by imposing an arbitrary distribution on the latent representation of the autoencoder. We discussed how this method can be extended to semi-supervised settings by incorporating the label information to better shape the hidden code distribution. Importantly, we demonstrated how it can be used to disentangle the style and label information of a dataset (Kingma et al., 2014; Cheung et al., 2014). Finally we showed that adversarial autoencoders can achieve state-of-the-art likelihoods on real-valued MNIST and Toronto Face datasets.

ACKNOWLEDGMENTS

We would like to thank Ilya Sutskever, Oriol Vinyals, Jon Gauthier, Sam Bowman and other members of the Google Brain team for the valuable comments. We thank the developers of TensorFlow (Abadi et al., 2015), a machine learning framework that allowed us to easily develop a fast and optimized code for GPU.

REFERENCES

Abadi, Martín, Agarwal, Ashish, Barham, Paul, Brevdo, Eugene, Chen, Zhifeng, Citro, Craig, Corrado, Greg S., Davis, Andy, Dean, Jeffrey, Devin, Matthieu, Ghemawat, Sanjay, Goodfellow, Ian, Harp, Andrew, Irving, Geoffrey, Isard, Michael, Jia, Yangqing, Jozefowicz, Rafal, Kaiser, Lukasz, Kudlur, Manjunath, Levenberg, Josh, Mané, Dan, Monga, Rajat, Moore, Sherry, Murray, Derek, Olah, Chris, Schuster, Mike, Shlens, Jonathon, Steiner, Benoit, Sutskever, Ilya, Talwar, Kunal, Tucker, Paul, Vanhoucke, Vincent, Vasudevan, Vijay, Viégas, Fernanda, Vinyals, Oriol, Warden, Pete, Wattenberg, Martin, Wicke, Martin, Yu, Yuan, and Zheng, Xiaoqiang. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

Bengio, Yoshua, Mesnil, Grégoire, Dauphin, Yann, and Rifai, Salah. Better mixing via deep representations. *International Conference on Machine Learning (ICML)*, 2013.

Bengio, Yoshua, Thibodeau-Laufer, Eric, Alain, Guillaume, and Yosinski, Jason. Deep generative stochastic networks trainable by backprop. *International Conference on Machine Learning (ICML)*, 2014.

Buades, A., Coll, B., and Morel, J. M. A review of image denoising algorithms, with a new one. *Simul*, 4:490–530, 2005.

Burda, Yuri, Grosse, Roger, and Salakhutdinov, Ruslan. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015.

Cheung, Brian, Livezey, Jesse A, Bansal, Arjun K, and Olshausen, Bruno A. Discovering hidden factors of variation in deep networks. *arXiv preprint arXiv:1412.6583*, 2014.

Goodfellow, Ian, Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pp. 2672–2680, 2014.

Hinton, Geoffrey E., Osindero, Simon, and Teh, Yee Whye. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

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

Kingma, Diederik P, Mohamed, Shakir, Rezende, Danilo Jimenez, and Welling, Max. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, pp. 3581–3589, 2014.

Li, Yujia, Swersky, Kevin, and Zemel, Richard. Generative moment matching networks. *International Conference on Machine Learning (ICML)*, 2015.

Lyu, S and Simoncelli, E P. Modeling multiscale subbands of photographic images with fields of Gaussian scale mixtures. *IEEE Trans. Patt. Analysis and Machine Intelligence*, 31(4):693–706, Apr 2009. ISSN 0162-8828. doi: 10.1109/TPAMI.2008.107.

Netzer, Yuval, Wang, Tao, Coates, Adam, Bissacco, Alessandro, Wu, Bo, and Ng, Andrew Y. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, pp. 5. Granada, Spain, 2011.

Rezende, Danilo Jimenez, Mohamed, Shakir, and Wierstra, Daan. Stochastic backpropagation and approximate inference in deep generative models. *International Conference on Machine Learning*, 2014.

Roweis, Sam T. and Saul, Lawrence K. Nonlinear dimensionality reduction by locally linear embedding. *SCIENCE*, 290:2323–2326, 2000.

Salakhutdinov, Ruslan and Hinton, Geoffrey E. Deep boltzmann machines. In *International Conference on Artificial Intelligence and Statistics*, pp. 448–455, 2009.