When Variational Auto-encoders meet Generative Adversarial Networks

Jianbo Chen Billy Fang Cheng Ju

14 December 2016

Abstract

Variational auto-encoders are a promising class of generative models. In this project, we first review variational auto-encoders (VAE) and its popular variants including the Conditional VAE [17], VAE for semi-supervised learning [10] and DRAW [2]. We also investigate various applications of variational auto-encoders. Finally, we propose models which combine variants of VAE with another popular deep generative model: generative adversarial networks [6], generalizing the idea proposed in [12].

1 Introduction

Unsupervised learning from large unlabeled datasets is an active research area. In practice, millions of images and videos are unlabeled and one can leverage them to learn good intermediate feature representations via approaches in unsupervised learning, which can then be used for other supervised or semi-supervised learning tasks such as classification. One approach for unsupervised learning is to learn a generative model. Two popular methods in computer vision are variational auto-encoders (VAEs) [11] and generative adversarial networks (GANs) [6].

Variational auto-encoders are a class of deep generative models based on variational methods. With sophisticated VAE models, one can not only generate realistic images, but also replicate consistent "style." For example, DRAW [7] was able to generate images of house numbers with number combinations not seen in the training set, but with a consistent style/color/font of street sign in each image. Additionally, as models learn to generate realistic output, they learn important features along the way, which potentially can be used for classification; we consider this in the Conditional VAE and semi-supervised learning models. However, one main criticism of VAE models is that their generated output is often blurry.

Generative adversarial networks are another class of deep generative models. GANs introduce a game theoretic component to the training process, in which a discriminator learns to distinguish real images from generated images, while at the same time the generator learns to fool the discriminator.

In this report, we make the following contributions:

- We do a survey on the currently popular variants of variational auto-encoders and implement them in TensorFlow.
- Based on previous work [12], we combine popular variants of variational auto-encoders with generative adversarial networks, addressing the problem of blurry images.
- We investigate a few applications of generative models, including image completion, style transfer and semi-supervised learning.

On the one hand, we were able to generate less blurry images by combining the Conditional VAE with a GAN, compared to the original Conditional VAE [17]. On the other hand, we did not get comparable accuracy in semi-supervised learning when we tried to combine VAE with GAN. DRAW yields strange results when combined with GAN. These problems and their potential solutions are detailed in section 5.

2 Overview of VAE models

2.1 Variational auto-encoders

In the **variational auto-encoder (VAE)** literature, we consider the following latent variable model for the data x.

$$\begin{aligned} z &\sim \mathcal{N}(0, I) \\ x \mid z &\sim f(x; z, \theta). \end{aligned}$$

Here, $f(x; z, \theta)$ is some distribution parameterized by z and θ. In our work, we mainly consider $x \mid z$ being [conditionally] independent Bernoulli random variables (one for each pixel) with parameters being some function of the latent variable z and possibly other parameters θ.

We omit motivation of the variational approach, which is explained in more detail in [3]. We consider the following lower bound, valid for any distribution $Q(z \mid x)$.

$$\log p(x) \geq \mathbb{E}_{z \sim Q(\cdot \mid x)}[\log p(x \mid z)] - \mathrm{KL}(Q(z \mid x) \| p(z)) =: -\mathcal{L}(x). \tag{1}$$

In variational inference, we would maximize the right-hand side over a family of distributions $Q(z \mid x)$. Here we restrict our attention to $Q(z \mid x)$ being Gaussian with some mean $\mu(x)$ and diagonal covariance $\Sigma(x)$ being functions (namely, neural networks) of x. For the VAE framework however, we also learn $p(x \mid z) = f(x; z, \theta)$; this too, is a neural network.

The VAE model is trained by maximizing the right-hand side of (1), and the structure is depicted in Figure 1. The first neural network, called the encoder, takes in the data x and outputs the mean $\mu(x)$ and variance $\Sigma(x)$ of $Q(z \mid x)$, which then can be used to compute the KL-divergence term in the loss. We then sample $z \sim Q(z \mid x)$ by computing $\Sigma(x) \cdot \epsilon + \mu(x)$ where $\epsilon \sim \mathcal{N}(0, I)$, and pass z through the second neural network, called the decoder, which allows us to compute $p(x \mid z)$. Since $z \sim Q(z \mid x)$, this is an unbiased estimate of $\mathbb{E}_{z \sim Q(\cdot \mid x)}[\log p(x \mid z)]$.

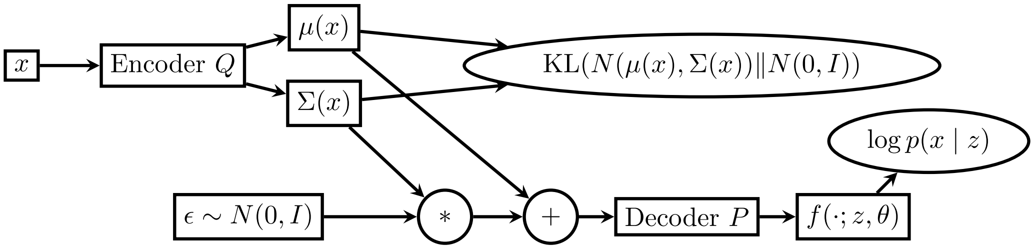

Figure 1: Variational auto-encoder structure.

To generate new data, we simply sample $z \sim \mathcal{N}(0, I)$ and input it into the trained decoder. More information can be found in the original paper [11] as well as in the tutorial [3].

2.2 Conditional variational auto-encoders

One variant of the VAE is the **Conditional variational auto-encoder (CVAE)** [17], which makes use of side information y, such as a label or part of an image. Mathematically the variational approach is the same as above except with conditioning on y.

$$\log p(x \mid y) \geq \mathbb{E}_{z \sim Q(\cdot \mid y, x)}[\log p(x \mid y, z)] - \mathrm{KL}(Q(z \mid x, y) \| p(z \mid y)) =: -\mathcal{L}(x, y).$$

The CVAE structure appears in Figure 2. Structurally, the only difference is that both the encoder and decoder take the side information y as extra input.

At generation time, we input not only a fresh sample $z \sim \mathcal{N}(0, I)$, but also some side information y into the decoder. We explore different examples of side information in Section 4.4.

2.3 Semi-supervised learning

Kingma et al. [10] introduced a variant of the VAE framework tailored for semi-supervised learning (SSL), where the goal is classification after training on a dataset that has both labeled and unlabeled examples. Unlike

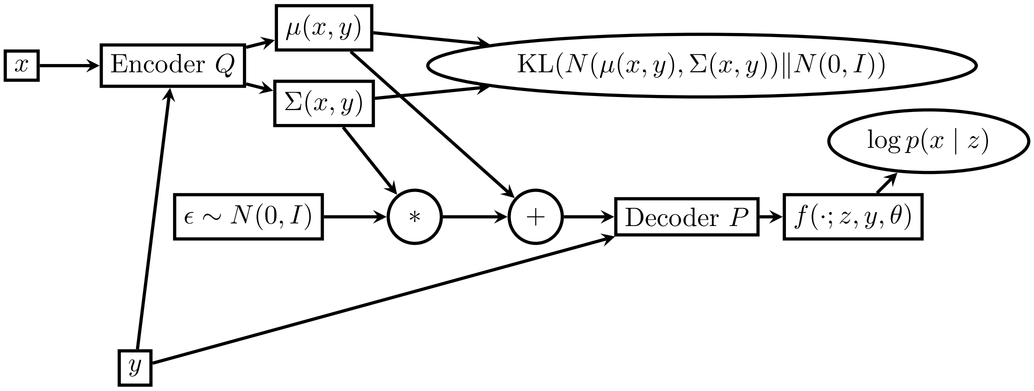

Figure 2: Conditional variational auto-encoder structure

the CVAE where everything was conditioned on the label y, the SSL model gives y a categorical distribution. The generative model (called "M2" in [10]) is as follows.

$$\begin{aligned} y &\sim \mathrm{Cat}(\pi) \\ z &\sim \mathcal{N}(0,I) \\ x \mid y,z &\sim f(x; y, z, \theta) \end{aligned}$$

We maximize a lower bound on the log likelihood (of both the data x and the label y) over normal distributions $Q(z \mid x,y)$ and categorical distributions $Q(y \mid x)$. However, for unlabeled examples, we marginalize over $y \sim q(y \mid x)$. This gives us the labeled and unlabeled losses $\mathcal{L}(x,y)$ and $\mathcal{U}(x)$ below.

$$\begin{aligned} \log p(x,y) &\geq \mathbb{E}_{z \sim Q(z|x,y)}[\log p(x \mid y,z) + \log p(y)] - \mathrm{KL}(Q(z \mid x,y)\|p(z)) =: -\mathcal{L}(x,y) \\ \log p(x) &\geq \sum_y Q(y \mid x)(-\mathcal{L}(x,y)) + H(Q(y \mid x)) =: -\mathcal{U}(x). \end{aligned}$$

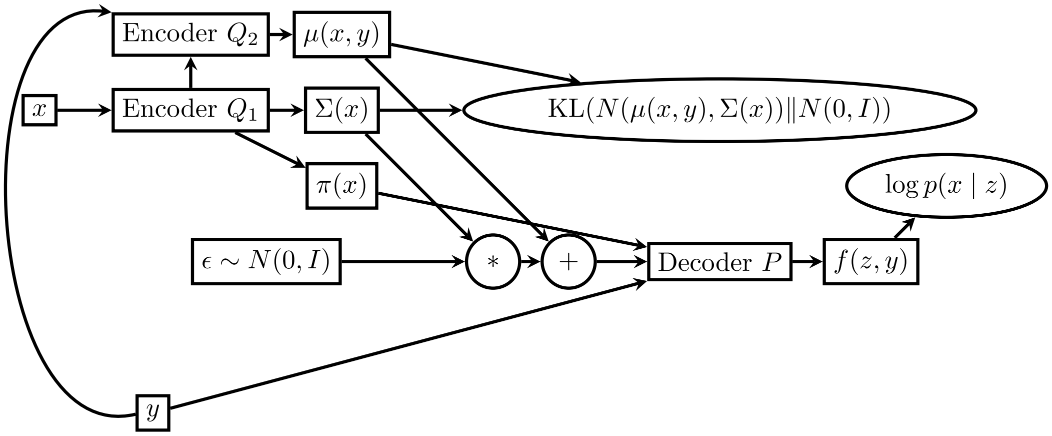

Figure 3: Model for semi-supervised learning

The total loss is

$$\sum_{\text{labeled } (x,y)} \mathcal{L}(x,y) + \sum_{\text{unlabeled } x} \mathcal{U}(x).$$

In [10], they also add an extra classification loss $\alpha \mathbb{E}_{\text{labeled } (x,y)}[-\log Q(y \mid x)]$ so that the predictive distribution $q(y \mid x)$ also contributes to $\mathcal{L}(x,y)$.

We can use this for labeled image generation as before: input $z \sim \mathcal{N}(0,I)$ and y into the decoder.

For classification, we input the unlabeled image into the decoder and return the class with the highest probability according to the categorical distribution parameterized by $\pi(x)$.

2.4 DRAW

The Deep Recurrent Attentive Writer (DRAW) [7] is an extension of the vanilla variational auto-encoder generator. It applies the sequential variational auto-encoding framework to iteratively construct complicated images, with a soft attention mechanism. There are three main differences between the DRAW and vanilla VAE.

1. Both the encoder and decoder of DRAW are recurrent networks.
2. The decoders outputs are successively added to the distribution that will ultimately generate the data (image), as opposed to emitting this distribution in a single step.
3. A dynamically updated attention mechanism is used in DRAW to restrict the input region observed by the encoder, which is computed by the previous decoder.

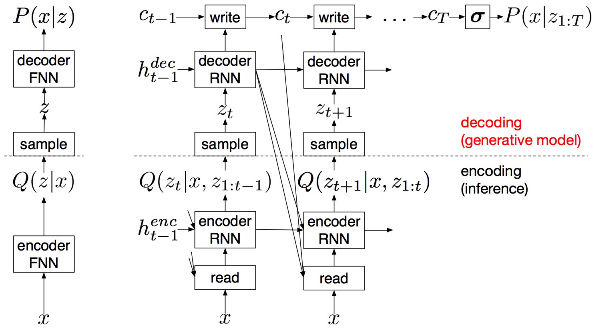

Figure 4: Conventional Variational Auto-Encoder (left) and DRAW Network [7] (right)

Figure 4 shows the structure of DRAW. The input for each encoder cell is convoluted image, where the convolution is computed by the previous decoder cell. Thus the input for each encoder is dynamically computed.

The loss comes from two parts. The decoder loss is the same as the vanilla VAE: only the final canvas matrix (which is the cumulative sum of the previous decoder output) is used to compute the distribution of the data $D(x|c_T)$. If the input is binary, the natural choice for D is a Bernoulli distribution with means given by $\sigma(c_T)$, and the reconstruction loss is the negative log probability of x under D:

$$L^x = -\log D(x|c_T)$$

The encoder loss is different. DRAW uses cumulative Kullback-Leibler divergence across time T as the encoder loss:

$$L^z = \sum_{t=1}^{T} KL(Q(Z_t|h_t^{enc})||Z_t) = \frac{1}{2}\sum_{t=1}^{T}(\mu_t^2 + \sigma_t^2 - \log \sigma_t^2) - T/2$$

The total loss L for the network is the expectation of the sum of the reconstruction and latent losses:

$$L = \mathbb{E}_{z\sim Q}(L^z + L^x).$$

Conditional DRAW is an extension of vanilla DRAW. The input of the encoder and decoder of conditional DRAW is the same as vanilla DRAW, except the additional information (e.g. the label of the image).

3 Combining generative adversarial networks with VAEs

In this section, we investigate how we can combine various variants of VAEs reviewed above with another popular generative model: generative adversarial networks (GANs) [6]. In a GAN, a generator and a discriminator are trained together. The discriminator is trained to distinguish “real” images from “fake” (i.e. generated), while the generator is trained to generate images that the discriminator will classify as “real.” Abstractly, combining the GAN framework with VAEs is straightforward: the generator in the GAN framework is the VAE. Below we discuss the details and our results for various variants of VAEs.

3.1 VAE with GAN

The basic idea is the following: Let $D(x)$ denote the output of the discriminator for an image x; this represents the probability that x is a real image. As in GAN, we train the generator (which is a VAE here) and the discriminator alternately, by minimizing two losses separately.

The generator loss is formed by adding an extra term $-\log D(\mathrm{Dec}(z))$, which punishes the generated images being classified as fake. The discriminator loss is $-\log D(x) - \log(1 - D(\mathrm{Dec}(z))$, the same as in a pure GAN.

Based on ideas in [12], we make the following two modifications on the simplified model:

- Replacing the VAE decoder/reconstruction loss $\mathbb{E}_{z\sim Q(\cdot|x)} \log p(x \mid z)$ with a dissimilarity loss $\mathcal{L}_{llike}^{\mathrm{Dis}_l}$ based on the lth layer inside the discriminator. That is, we input both the original image x and a generated image $\mathrm{Dec}(z)$ into the discriminator, and use the L^2 distance between the output of the lth layer of the discriminator as the decoder loss (this can be interpreted as maximizing the log likelihood of a Gaussian distribution) instead of the original pixel-based loss. The intuition is that the deeper layers of the discriminator will learn good features of the images, and that using this as the decoder/reconstruction loss will be more informative than using a pixel-based loss. Mathematically, the reconstruction loss is a l_2 loss:

  $$\mathcal{L}_{llike}^{\mathrm{Dis}_l} = \|f_l(x) - f_l(\widetilde{x})\|^2/2\sigma^2 + Const.$$

  where $f_l(x)$ is the result of x passing through the lth layer of the discriminator network, and $\widetilde{x}$ is the result getting from letting x pass through the encoder and then the decoder.

- Incorporate the classification of the reconstructed image $\mathrm{Dec}(\mathrm{Enc}(x))$ by adding $-\log D(\mathrm{Dec}(\mathrm{Enc}(x)))$ and $-\log(1 - D(\mathrm{Dec}(\mathrm{Enc}(x)))))$ to the VAE and discriminator losses respectively. Since the reconstructed image is close to the original image, this will help train the discriminator.

In summary, the generator loss is

$$L_G = \mathcal{L}_{KL} + \mathcal{L}_{llike}^{\mathrm{Dis}_l} - \log D(\mathrm{Dec}(z)) - \log D(\mathrm{Dec}(\mathrm{Enc}(x))) \tag{2}$$

where z is generated from the prior $p(z)$ of latent variables ($p(z) = \mathcal{N}(0, I)$) and $\mathcal{L}_{KL} = KL(q(z|x), p(z))$ is the same as the regularization term in VAE. The discriminator loss is

$$L_D = -\log D(x) - \log(1 - D(\mathrm{Dec}(z)) - \log(1 - D(\mathrm{Dec}(\mathrm{Enc}(x)))), \tag{3}$$

where z is generated from the prior of latent variables ($N(0, I)$).

The structure of the model is shown in Figure 5 [12].

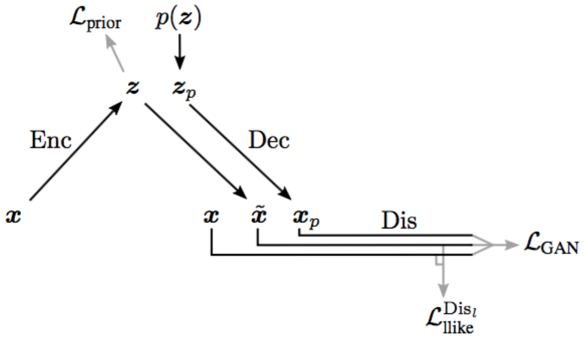

Figure 5: VAE with GAN model from [12]

3.2 CVAE with GAN

One generalization of VAE with GAN is to incorporate the side information y, such as a label or part of an image. Our idea is to put a GAN on top of a Conditional VAE. To make full use of the side information y, the structure of the discriminator of GAN is also modified. Suppose y is a label which has k classes. Then the discriminator D outputs a distribution on $k + 1$ classes. We wish the discrimator to assign the correct label among

k classes for a true example (x, y) but assign generated images to the extra $k+1$th class. Formally, define D as a function from the image space to a distribution on $k+1$ classes:

$$\sum_{c=1}^{k+1} D_c(x) = 1, D_c(x) \geq 0.$$

We still train the generator and the discriminator alternately. The generator loss is formed as a sum of the loss of a Conditional VAE and the loss from GAN which is a cross entropy loss over the output of the discriminator which punishes generated images for being classified into a wrong class. The mathematical representation is shown below. Recall that the encoder and the decoder of a conditional VAE both take y as an input. The cross entropy loss takes as input a distribution p on classes and a label y, and outputs $-\log p(Y = y)$.

$$\begin{aligned} L_G = \mathcal{L}_{KL} &+ \mathcal{L}_{llike}^{\mathrm{Dis}_l} \\ &+ \mathit{CrossEntropy}(D(\mathrm{Dec}(z, y)), y) \\ &+ \mathit{CrossEntropy}(D(\mathrm{Dec}(\mathrm{Enc}(x, y), y), y), \end{aligned}$$

where z is generated from the prior $p(z)$ of latent variables. $\mathcal{L}_{KL}$ and $\mathcal{L}_{llike}^{\mathrm{Dis}_l}$ are exactly the same as those in the previous section, except that we also feed in y when generating images from the decoder.

The discriminator loss is a cross entropy loss over the output of the discriminator which punishes classifying a real image into a wrong class or classifying a generated image into the first k classes. Concretely, the discriminator loss is

$$\begin{aligned} L_D = \mathit{CrossEntropy}&(D(x), y) \\ &+ \mathit{CrossEntropy}(D(\mathrm{Dec(z, y)}), y_{fake}) \\ &+ \mathit{CrossEntropy}(D(\mathrm{Dec(Enc(x, y), y)}), y_{fake}), \end{aligned}$$

where z is generated from the prior of latent variables and y_{fake} is the label for the extra $(k+1)$st class.

3.3 SSL with GAN

The VAE tailored for semi-supervised learning (SSL) can also be incorporated with a $k+1$-class GAN in a similar way as CVAE. Recall that the VAE-SSL loss is divided into two parts: a loss for labeled data and a loss for unlabeled data. The method of incorporating GAN to VAE-SSL is exactly the same as that of CVAE for labeled data; refer to the previous section for details. For unlabeled data, recall that the VAE loss is

$$\sum_y Q(y \mid x)(-\mathcal{L}(x, y)) + H(Q(y \mid x)) =: -\mathcal{U}(x),$$

where a categorical distribution $Q(y|x)$ is introduced. We mimic the idea that when a label y is needed as an input, the loss is averaged over $Q(y|x)$. Mathematically:

$$\begin{aligned} L_{G,unobserved} = \mathcal{L}_{KL} + H(Q(y|x)) + \sum_y Q(y|x)\Big(&\mathcal{L}_{llike}^{\mathrm{Dis}_l} \\ &+ \mathit{CrossEntropy}(D(\mathrm{Dec}(z, y)), y) \\ &+ \mathit{CrossEntropy}(D(\mathrm{Dec}(\mathrm{Enc}(x, y), y), y)\Big), \end{aligned}$$

where we emphasize that $\mathcal{L}_{llike}^{\mathrm{Dis}_l}$ also depends on y and needs to be averaged, and that the entropy term $H(Q(y|x))$ is introduced to regularize over Q as before. The discriminator loss is again averaged over $Q(y|x)$:

$$\begin{aligned} L_D = \sum_y Q(y|x)\big(&\mathit{CrossEntropy}(D(x), y) \\ &+ \mathit{CrossEntropy}(D(\mathrm{Dec(z, y)}), y_{fake}) \\ &+ \mathit{CrossEntropy}(D(\mathrm{Dec(Enc(x, y), y)}), y_{fake})\big). \end{aligned}$$

3.4 DRAW with GAN

Unlike the VAE, which only generates one image, DRAW generates a sequence of images. Thus, there are two potential way to combine DRAW with GAN: we could either add discriminator on the top of all the temporary canvas, or only on the top of the last output. In this project, we only consider the latter one. If we consider DRAW as a black-box generator like the vanilla VAE, the structure is exactly the same as VAE-GAN. The two losses that we minimize alternately take the same forms as (2) and (3), but instead use the decoder and the encoder from DRAW.

4 Experiments and Applications

In the project, we implemented all the models mentioned above and studied their performance. The code will be made available at https://github.com/Jianbo-Lab/vae-gan. In this section, we discuss details about the implementation, investigate applications of various models and compare the performances of various models. All models were trained with mini-batch stochastic gradient descent (SGD) with a mini-batch size of 100. All weights were initialized from a zero-centered Normal distribution. Batch-normalization [16] was added to all fully connected and convolutional layers except the output layers. The Adam [9] optimizer with default hyperparameters in TensorFlow was used for optimization. For various VAEs without GANs, we used the learning rate 0.001 while for all VAEs with GANs, we use learning rate 0.0001. Due to the page limit, we omit the detailed description of our implementations of VAE-GAN. We refer interested readers to our GitHub repository for details.

4.1 Tools

We implemented all the models in TensorFlow. We used a GeForce GTX 770 GPU through Amazon for training. Git and Github were also very helpful for collaboration and managing our codebase.

4.2 Dataset

We used the MNIST handwritten digits dataset [13] through TensorFlow as well as the Street View House Numbers (SVHN) dataset [15]. The MNIST dataset consists of a training dataset with $55,000$ labeled handwritten digits, a validation dataset with $5,000$ examples and a test set with $5,000$ examples. SVHN is a real-world image dataset similar in flavor to MNIST (e.g., the images are of small cropped digits), but incorporates an order of magnitude more labeled data (over $600,000$ digit images) and comes from a significantly harder, unsolved, real world problem (recognizing digits and numbers in natural scene images) [15]. We treated the MNIST images as a 28×28-long vector with pixel values in $[0, 1]$. We treated the SVHN images as a $32 \times 32 \times 3$-long vector with RGB values in $[0, 1]$. No other data cleaning was needed.

4.3 CVAE v.s. CVAE-GAN

This experiment aims to compare the performance of CVAE with that of CVAE-GAN in terms of generating images. In this experiment, we feed into a trained decoder of CVAE a label (1 to 10) and a randomly generated z from the standard normal $\mathcal{N}(0, I)$. Then the corresponding handwritten digits will be generated according to the label. Figure 6 shows ten generated samples of each digit after training on MNIST and SVHN using CVAE. Figure 7 shows ten generated samples of each digit after training on MNIST and SVHN using CVAE with GAN.

The networks of CVAE and CVAE-GAN are constructed based on structures for DCGAN [1] with slight modification. In particular, we replace pooling by strided convolutions (discriminator) and fractional-strided convolutions (decoder) and remove all unnecessary fully connected layers (except at the outputs). Below we briefly describe the network structure of the encoder, the decoder and the discriminator (for MNIST), following notations used in class [4]. The discriminator network is only used in CVAE-GAN while the encoder and the decoder networks are used both in CVAE and CVAE-GAN. Note that we use the same network structure for SVHN, except that the number of neurons at each layer increases by 8/7.

- Encoder:

$$\left\{ \begin{array}{r} \text{Input } x : [28 \times 28] \to \text{conv5-28} \to \text{conv5-56} \quad \to \text{conv5-108}\} \\ \text{Input } y : [10] \end{array} \right\} \to \left\{ \begin{array}{ll} \text{FC-50} & \to \Sigma \\ \text{FC-50} & \to \mu \end{array} \right\}$$

- Decoder:

$$\text{Input } z \text{ concat with } y : [50 + 10] \rightarrow \text{FC-7*7*128} \rightarrow \text{reshape}[7 \times 7 \times 128] \rightarrow$$
$$\rightarrow \text{deconv5-64} \rightarrow \text{deconv5-32} \rightarrow \text{output } x : [28 \times 28]$$

  Here deconv is also called fractionally-strided convolutions in literature [1].

- Discriminator:

$$\text{Input } x : [28 \times 28] \rightarrow \text{conv3-16} \rightarrow \text{conv3-32} \rightarrow \text{conv3-64} \rightarrow \text{FC-512} \rightarrow \left\{ \begin{array}{l} \text{FC-500} \rightarrow \text{Softmax} \\ l\text{th layer } f_l(x) \end{array} \right\}$$

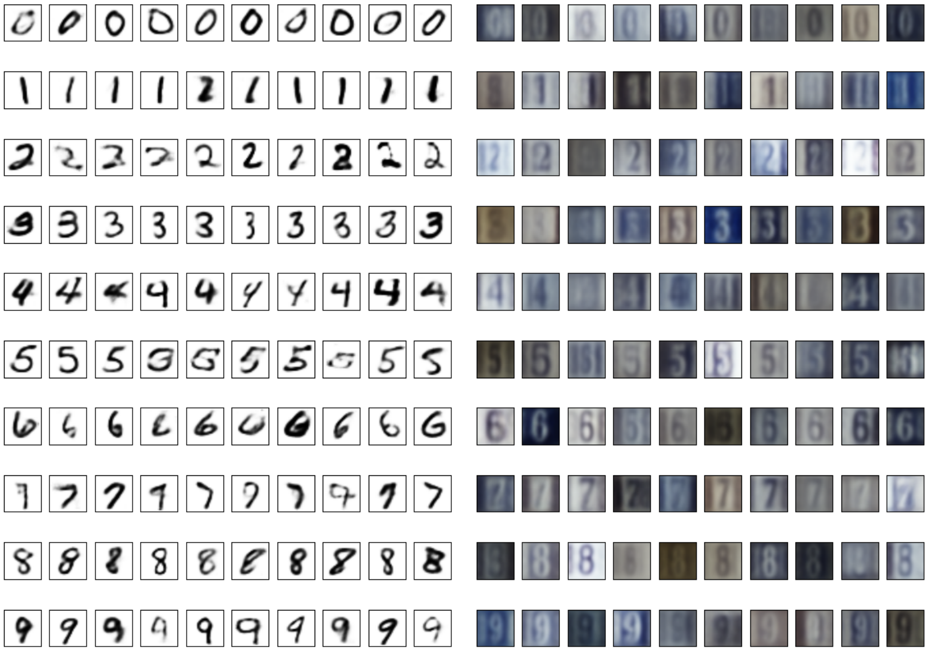

Figure 6: Purely generated images for MNIST (left) and SVHN (right) by CVAE. Since the CVAE incorporates labels into generation, we are able to generate a particular digit we choose.

Comparing the results of CVAE and CVAE-GAN, we can see CVAE-GAN generates less blurry MNIST images compared to CVAE. But for the digit "2," CVAE-GAN generates highly similar and strange images constantly. For the SVHN dataset, most of images generated by CVAE-GAN are less blurry, but there are also some weird images generated, like the digit "7" in the fourth column. We put the fixing of these problems to future work.

4.4 CVAE applications

When implementing the CVAE model, we also investigated various applications of CVAE. In particular, we show here how we use a simple fully-connected CVAE for image completion and style transfer.

4.4.1 Image completion

The side information y need not be a label. Here, we considered y as being the left half of an image. At generation time, we can input a new half-image into the decoder, and it will attempt to complete the image based on what it has learned from the training data. Our demonstration appears in Figure 8.

4.4.2 Style transfer

The CVAE can be used to transfer style from one image to the generated output. To do so, we input the base image (whose style we want to copy) into the encoder and obtain its latent "code" z. We then input this sampled

SVHN.

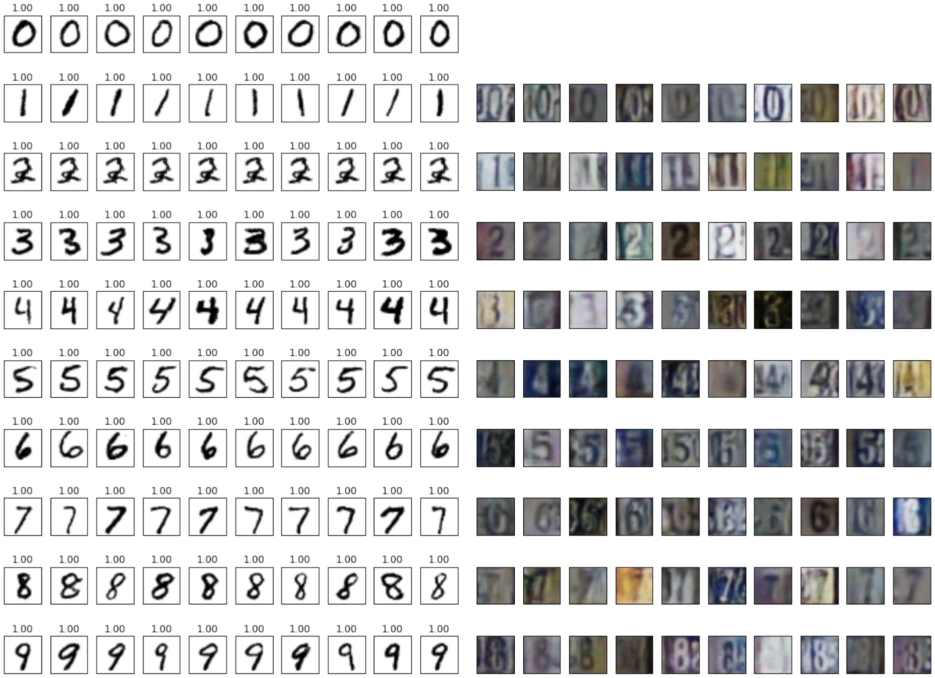

Figure 7: Purely generated images for MNIST (left) and SVHN (right) by CVAE-GAN.

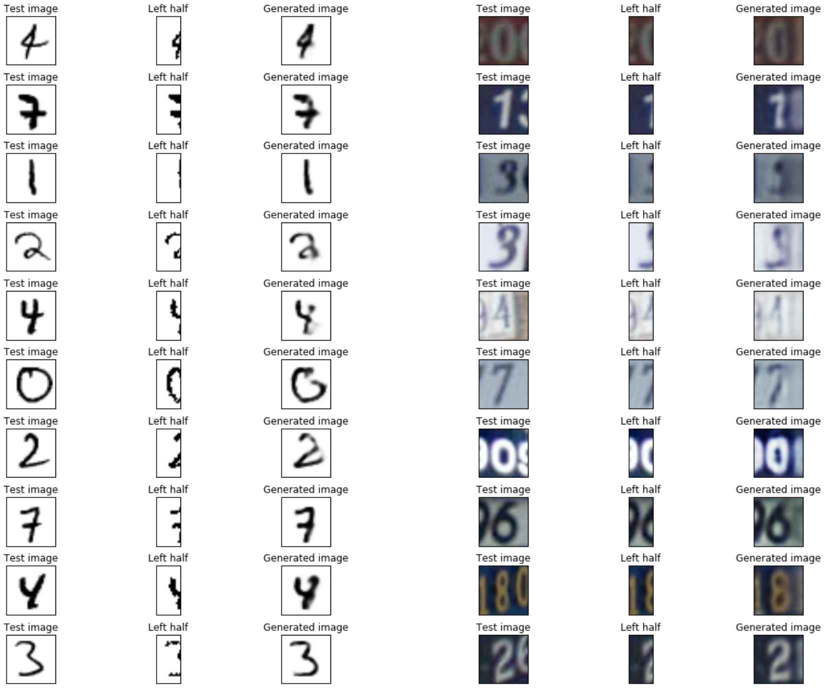

Figure 8: Image completion for MNIST (left) and SVHN (right). We take half-images from the test data and input them into the decoder, which attempts to complete the image based on what it has learned from the training data.

z, which captures the features and style of the base image, along with an arbitrary label y into the decoder. The output is then a digit with label y, but with the style of the original image. As shown in Figure 9, we are able to copy boldness and slant in the case of MNIST, as well as font style, font color, and background color in the case of SVHN.

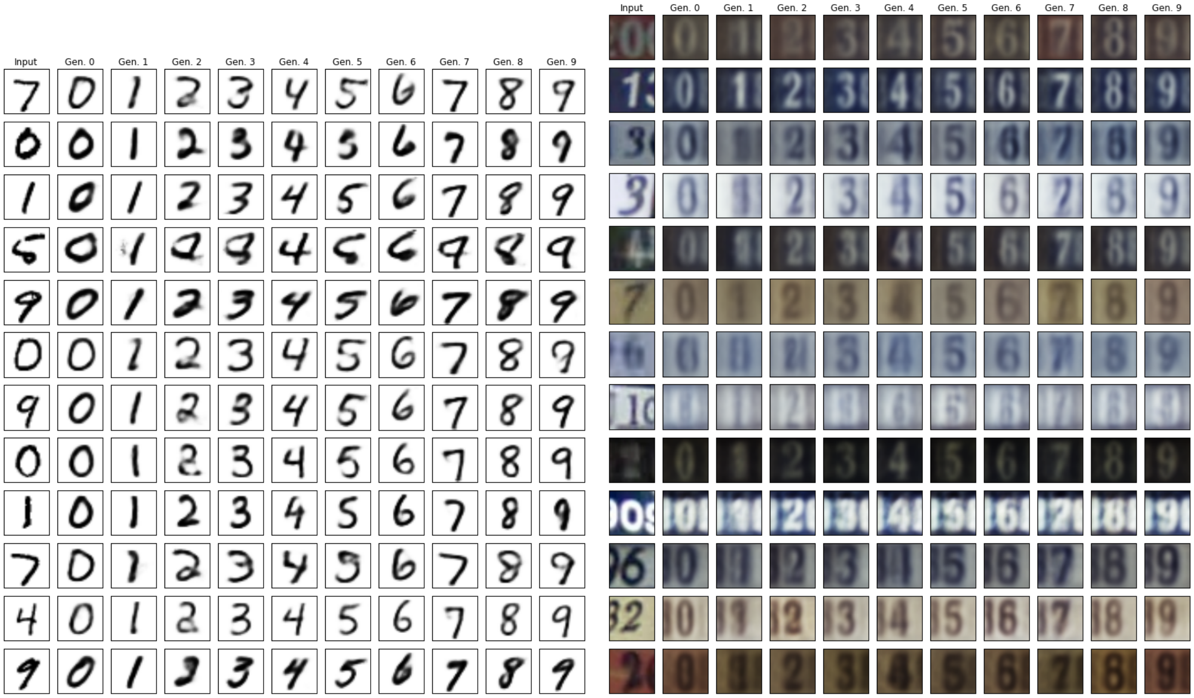

Figure 9: Style transfer for MNIST (left) and SVHN (right). We transfer the style of the first column to all digits 0 to 9.

4.5 Semi-supervised learning

This experiment compares the performance of VAE and that of VAE-GAN in semi-supervised learning. We followed the "M2" model of [10] depicted in Figure 3.

We took the 55000 examples in the MNIST dataset and retained only 1000 or 600 of the labels (so that more than 98% of the data is unlabeled!), with each label equally represented.

We considered both a fully-connected architecture and a convolutional architecture. For the discriminator of GAN, we still use the same structure as that of CVAE-GAN. Interested readers can refer to the previous section for details. Below we describe the network structures of the encoder and the decoder.

- Fully connected
  - In the fully connected model described below, each hidden layer has batch normalization and softplus activation.
  - Encoder:

$$\text{Input } x\text{: } [28 \times 28] :\to \text{FC-1024} \to \text{FC-1024} \to \begin{cases} h_1 \\ \text{FC-50} \to \Sigma(x) \\ \text{FC-50} \to \pi(x) \end{cases}$$

$$h_1 \text{ concat with } y : [1024 + 10] \to \text{FC-1024} \to \text{FC-50} \to \mu(x, y)$$

  - Decoder:

$$\text{Input } z \text{ concat with } y\text{: } [50 + 10] \to \text{FC-1024} \to \text{FC-1024} \to \text{FC-28*28-sigmoid} \to \text{output } x\text{: } [28 \times 28]$$

- Convolutional model
  - For the convolutional version we replaced the hidden fully-connected layers with 5×5 convolutions (in the encoder) and convolution transposes ("deconvolutions"). Below, all convolutions and deconvolutions have stride 1. The argument of Conv and Deconv is the number of filters. 2×2 max pooling and 2×2 upsampling are denoted by $\Downarrow$ and $\Uparrow$ respectively.

- All layers have batch normalization except the output layers. All fully-connected layers have ReLU activation except the output layers.

$$\begin{aligned}
Q_1 :&\quad x, \mathrm{Conv}(20), \Downarrow, \mathrm{Conv}(50), \Downarrow, \mathrm{FC}(1024), \{\text{FC for } \Sigma,\ \text{FC-sigmoid for } \pi\} \\
Q_2 :&\quad (x, y), \mathrm{FC}(1024), \mathrm{FC} \\
P :&\quad z, \mathrm{FC}(500), \mathrm{FC}(7 \times 7 \times 50), \Uparrow 2, \mathrm{Deconv}(20), \Uparrow, \mathrm{Deconv}(1), \\
&\qquad \text{concat with } y, \mathrm{FC}(512), \text{FC-sigmoid}
\end{aligned}$$

- Encoder:

$$\text{Input } x\text{: } [28 \times 28] :\rightarrow \text{conv5-20} \rightarrow\Downarrow\rightarrow \text{conv5-50} \rightarrow\Downarrow\rightarrow \text{FC-1024} \rightarrow \begin{cases} h_1 \\ \text{FC-50} \rightarrow \Sigma(x) \\ \text{FC-50} \rightarrow \pi(x) \end{cases}$$

$$h_1 \text{ concat with } y : [1024 + 10] \rightarrow \text{FC-1024} \rightarrow \text{FC-50} \rightarrow \mu(x, y)$$

- Decoder:

$$\left.\begin{array}{r} \text{Input } z\text{: } [50] \rightarrow \text{FC-500} \rightarrow \text{FC-7*7*50} \rightarrow\Uparrow\rightarrow \text{deconv5-20} \rightarrow\Uparrow\rightarrow \text{deconv5-1} \\ \text{Input } y\text{: } [10] \end{array}\right\}$$

$$\rightarrow \text{FC-512} \rightarrow \text{FC-28*28-sigmoid} \rightarrow \text{output } x\text{: } [28 \times 28]$$

Table 1 shows the resulting classification error on the test dataset. Currently we are unable to perform as well as the original paper [10]. Some potential reasons are that we have pooling after convolutional layers, we are not using the most appropriate activation functions in the discriminator and in the generator and there are fully connected layers in the network. We are currently working on new network structures that replace pooling by strided convolutions (discriminator) and fractional-strided convolutions (decoder), repace ReLU in the discriminator by LeakyReLU and remove unnecessary fully connected layers (as suggested in [1]).

	1000 labeled	600 labeled
Fully connected	5.1%	12.0%
Convolutional	4.8%	6.2%
Convolutional with GAN	8.1%	9.2%
Kingma et al. [10]	2.4%	2.6%

Table 1: Validation/test error on MNIST (55000 training examples)

4.6 DRAW v.s. DRAW-GAN

This experiment compares the performance of DRAW and DRAW-GAN.

- Encoder: a standard LSTM cell. The size of the hidden state is 256.
- Decoder: a standard LSTM cell. The size of the hidden state is 256.
- Discriminator (for DRAW-GAN): we use the same discriminator as CVAE. Due to the limited space, we only report the experimental results for vanilla DRAW and DRAW-GAN (omitting conditional DRAW results). In addition, replacing the decoder loss with GAN layer loss did not give satisfactory performance. Thus we used the reconstruction loss on the original image (the L2 or cross-entropy loss between original image and reconstructed image).

$$\text{Input } x : [28 \times 28] \rightarrow \text{conv3-16} \rightarrow \text{conv3-32} \rightarrow \text{conv3-64} \rightarrow \text{FC-512} \rightarrow \left\{ \begin{array}{l} \text{FC-500} \rightarrow \text{Softmax} \\ l\text{th layer } f_l(x) \end{array} \right\}$$

It is interesting to see how the style of the images generated by DRAW-GAN differ from the reconstructed images (Figure 11, right and left respectively). The reconstructed image is very similar to the real image, while the generated image is more winding than the real one.

We see that simply taking DRAW as a black box and adding a GAN on top of it directly does not produce good results. Some potential solutions include incorporating an attention mechanism in the GAN and investigating how to appropriately add a GAN to each image in the sequence generated by DRAW.

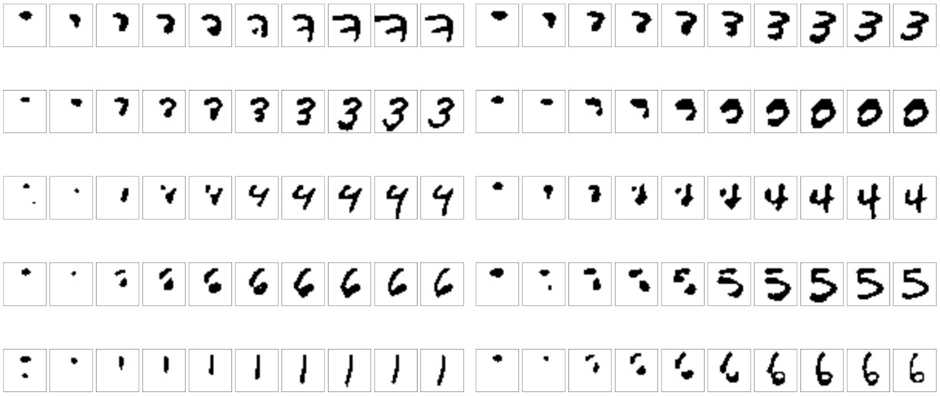

Figure 10: Images on the left are reconstructed by the real input images using DRAW. Images on the right are generated from random Gaussian input using DRAW.

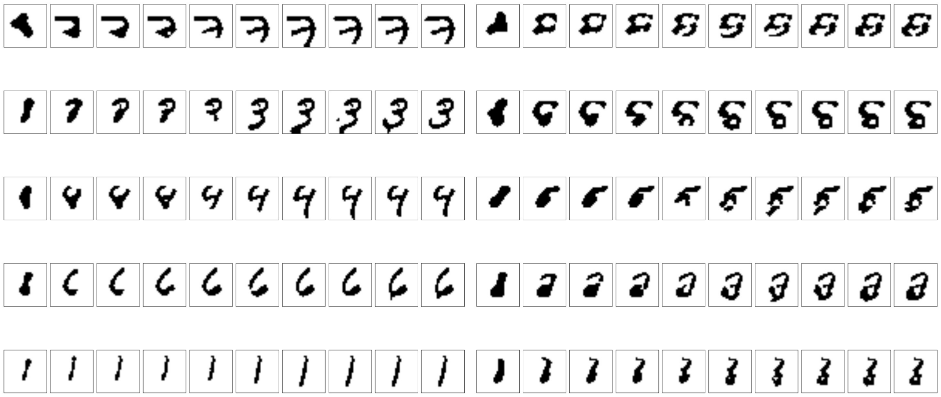

Figure 11: Images on the left are reconstructed by the real input images using DRAW-GAN. Images on the right are generated from random Gaussian input using DRAW-GAN.

5 Conclusion and Future Work

Variational auto-encoders and generative adversarial networks are two promising classes of generative models which have their respective drawbacks. For example, VAEs generate blurry images while GANs are difficult to train and generate images which lack variaty. This work serves as an initiate of combining various variants of VAEs with GANs. In particular, we propose CVAE-GAN, SSL-VAE-GAN and DRAW-GAN. We also apply generative models to the problems of image completion, style transfer and semi-supervised learning. While CVAE-GAN generated better images than the baseline model CVAE, SSL-VAE-GAN and DRAW-GAN fail to achieve expected performance. We propose several potential solutions to these problems and leave them to future work.

6 Lessons Learned

- TensorFlow, although tricky to learn at first, became quite helpful when building more complicated models. In particular, the TensorFlow-Slim library allowed us to access standard neural network layers easily so that we did not have to manage weights and variables manually.

- Some of the models were difficult to train, and batch normalization was critical to reduce sensitivity to hyperparameters (e.g. learning rate) and to help bring the loss down.

- The pooling layer is not helpful in generateive models of images like VAEs and GANs.

- Use a smaller learning rate (0.0001) to train GANs but use a higher one (0.001) to train VAEs.

7 Team Contributions

- Jianbo Chen (?%): implemented VAE framework on which all other implementations are based, improved convolutional architecture, improved/implemented GAN-based models, contributed to poster, wrote report
- Billy Fang (?%): implemented CVAE (with digit generation, image completion, and style transfer), implemented SSL, implemented baseline GAN-based models, contributed to poster, wrote report.
- Cheng Ju (?%): implemented and experimented with DRAW, conditional DRAW, DRAW with GAN. Build general DRAW object that has options to construct flexible combinations of DRAW variants (conditional/unconditional and vanilla/GAN)

References

[1] Soumith Chintala Alec Radford, Luke Metz. Unsupervised representation learning with deep convolutional generative adversarial networks. *ICLR*, 2016.

[2] Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. In *Advances in neural information processing systems*, pages 2980–2988, 2015.

[3] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[4] Justin Johnson Fei-Fei Li, Andrej Karpathy. Lecture notes of cs231n. *http://cs231n.github.io/*, 2016.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016.

[6] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Advances in Neural Information Processing Systems*, 2014.

[7] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. DRAW: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

[8] Xianxu Hou, Linlin Shen, Ke Sun, and Guoping Qiu. Deep feature consistent variational autoencoder. *arXiv preprint arXiv:1610.00291*, 2016.

[9] Diederik P Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. *arXiv reprint arXiv:1412.6980*, 2014.

[10] Diederik P. Kingma, Danilo Jimenez Rezende, Shakir Mohamed, and Max Welling. Semi-supervised learning with deep generative models. *CoRR*, abs/1406.5298, 2014.

[11] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[12] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. *arXiv preprint arXiv:1512.09300*, 2015.

[13] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[14] Marcus Liwicki and Horst Bunke. Iam-ondb-an on-line english sentence database acquired from handwritten text on a whiteboard. In *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, pages 956–961. IEEE.

[15] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[16] Christian Szegedy Sergey Ioffe. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Journal of Machine Learning Research*, 2016.

[17] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems*, pages 3483–3491, 2015.

[18] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*, 2015.