Generative Adversarial Network and its Variations

Presented by Yunchen Pu

September 30, 2016
GAN: Generative Adversarial Nets
DCGAN: Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks
ConGAN: Conditional Generative Adversarial Nets
GAN-T2I: Generative Adversarial Text to Image Synthesis
InfoGAN: InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets
AAE: Adversarial Autoencoders
VAE/GAN: Autoencoding beyond pixels using a learned similarity metric
Generative Model of Natural Image

- Generative Model:
  - Have training set: \( X \sim P_{data} \)
  - Learn a model: \( X \sim P_{model} \)
  - Samples drawn from \( P_{model} \) reflect structure of \( P_{data} \)
  - Samples from data distribution have high likelihood under \( P_{model} \)

- Unsupervised representation learning
  - Transfer learned representation to discriminative tasks, retrieval, clustering, etc.
  - Semi-supervised learning: very little labeled data, regularization, etc.

- Understand data; Density estimation ...
Generative Adversarial Networks

Basic Model

- A game between:
  - Discriminative model D
  - Generative model G

- G: trained to maximize the probability of D making a mistake

- D: trained to estimate the probability that a sample came from data distribution rather than G
Minimax value function:

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

- Discriminator pushes up
- Discriminator's ability to recognize data as being real
- Discriminator's ability to recognize generator samples as being fake

For $G$ fixed, the optimal discriminator $D$ is:

$$D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)}$$  (1)
Generative Adversarial Networks

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, \( k \), is a hyperparameter. We used \( k = 1 \), the least expensive option, in our experiments.

\[
\textbf{for} \text{ number of training iterations do}
\]
\[
\textbf{for} \ k \ \text{steps do}
\]
\[
\begin{align*}
&\bullet \text{ Sample minibatch of } m \text{ noise samples } \{z^{(1)}, \ldots, z^{(m)}\} \text{ from noise prior } p_g(z). \\
&\bullet \text{ Sample minibatch of } m \text{ examples } \{x^{(1)}, \ldots, x^{(m)}\} \text{ from data generating distribution } p_{\text{data}}(x).
\end{align*}
\]
\[
\text{ Update the discriminator by ascending its stochastic gradient:}
\]
\[
\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D \left( x^{(i)} \right) + \log \left( 1 - D \left( G \left( z^{(i)} \right) \right) \right) \right].
\]
\[
\textbf{end for}
\]
\[
\begin{align*}
&\bullet \text{ Sample minibatch of } m \text{ noise samples } \{z^{(1)}, \ldots, z^{(m)}\} \text{ from noise prior } p_g(z). \\
&\bullet \text{ Update the generator by descending its stochastic gradient:}
\end{align*}
\]
\[
\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log \left( 1 - D \left( G \left( z^{(i)} \right) \right) \right).
\]
\[
\textbf{end for}
\]

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.
**Generative Adversarial Networks**

**Experimental Results**

<table>
<thead>
<tr>
<th>MNIST</th>
<th>TFD</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="MNIST Samples" /></td>
<td><img src="image2" alt="TFD Samples" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CIFAR-10 (fully connected)</th>
<th>CIFAR-10 (convolutional)</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image3" alt="CIFAR-10 (fully connected) Samples" /></td>
<td><img src="image4" alt="CIFAR-10 (convolutional) Samples" /></td>
</tr>
</tbody>
</table>

**Figure**: The rightmost column shows the nearest training example of the neighboring sample.
- Use deep CNN for generator and discriminator instead of MLP.
  - Replace any pooling layers with strided convolution.
  - Use batchnorm in both the generator and the discriminator.
  - Remove fully connected hidden layers for deeper architectures.
  - Uses Tanh for the output (and sigmoid).
  - Use LeakyReLU in the discriminator and ReLU in the generator.
- Use the trained discriminators for image classification tasks.
Figure 2: Generated bedrooms after one training pass through the dataset. Theoretically, the model could learn to memorize training examples, but this is experimentally unlikely as we train with a small learning rate and minibatch SGD. We are aware of no prior empirical evidence demonstrating memorization with SGD and a small learning rate.

Figure 3: Generated bedrooms after five epochs of training. There appears to be evidence of visual under-fitting via repeated noise textures across multiple samples such as the base boards of some of the beds.

We use Imagenet-1k (Deng et al., 2009) as a source of natural images for unsupervised training. We train on 32 × 32 min-resized center crops. No data augmentation was applied to the images.
Figure 7: Vector arithmetic for visual concepts. For each column, the $Z$ vectors of samples are averaged. Arithmetic was then performed on the mean vectors creating a new vector $Y$. The center sample on the right hand side is produced by feeding $Y$ as input to the generator. To demonstrate the interpolation capabilities of the generator, uniform noise sampled with scale $\pm 0.25$ was added to $Y$ to produce the 8 other samples. Applying arithmetic in the input space (bottom two examples) results in noisy overlap due to misalignment. Further work is needed to tackle this form of instability. We think that extending this framework...
Figure: A "turn" vector was created from four averaged samples of faces looking left vs looking right. By adding interpolations along this axis to random samples we were able to reliably transform their pose.

Further investigations into the properties of the learnt latent space would be interesting as well.

ACKNOWLEDGMENTS

We are fortunate and thankful for all the advice and guidance we have received during this work, especially that of Ian Goodfellow, Tobias Springenberg, Arthur Szlam and Durk Kingma. Additionally we'd like to thank all of the folks at indico for providing support, resources, and conversations, especially the two other members of the indico research team, Dan Kuster and Nathan Lintz. Finally, we'd like to thank Nvidia for donating a Titan-X GPU used in this work.

REFERENCES


Coates, Adam and Ng, Andrew. Selecting receptive fields in deep networks. NIPS, 2011.


DCGAN
Image generation results

Figure: Generations of a DCGAN that was trained on the Imagenet-1k dataset.
DCGAN
Classification results

Table: Classification accuracy on CIFAR 10

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-layer K-means</td>
<td>80.6</td>
</tr>
<tr>
<td>3 Layer K-means</td>
<td>82.0</td>
</tr>
<tr>
<td>Exemplar CNN</td>
<td>84.3</td>
</tr>
<tr>
<td>DCGAN + SVM</td>
<td>82.8</td>
</tr>
</tbody>
</table>

Table: Classification error on SVHN with 1000 labels

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>KNN</td>
<td>77.93</td>
</tr>
<tr>
<td>TSVM</td>
<td>66.55</td>
</tr>
<tr>
<td>M1+M2</td>
<td>36.02</td>
</tr>
<tr>
<td>DCGAN + SVM</td>
<td>22.48</td>
</tr>
<tr>
<td>CNN</td>
<td>28.87</td>
</tr>
</tbody>
</table>
**Conditional Generative Adversarial Nets**

**Model**

- Condition generation on additional info $y$ (e.g. class label)
- Objective function:

$$\min_G \max_D \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x|y)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z|y)|y))]$$  \hspace{1cm} (2)

---

**Figure :** Model structure for conditional GAN

**Figure :** Conditional GAN with MLP
Conditional Generative Adversarial Nets
Image generation results on MNIST

Table 1: Parzen window-based log-likelihood estimates for MNIST. We followed the same procedure as [8] for computing these values.

The discriminator maps $x$ to a maxout [6] layer with 240 units and 5 pieces, and $y$ to a maxout layer with 50 units and 5 pieces. Both of the hidden layers mapped to a joint maxout layer with 240 units and 4 pieces before being fed to the sigmoid layer. (The precise architecture of the discriminator is not critical as long as it has sufficient power; we have found that maxout units are typically well suited to the task.)

The model was trained using stochastic gradient decent with mini-batches of size 100 and initial learning rate of $0.1$ which was exponentially decreased down to $0.000001$ with decay factor of $1.00004$. Also momentum was used with initial value of $0.5$ which was increased up to $0.7$. Dropout [9] with probability of 0.5 was applied to both the generator and discriminator. And best estimate of log-likelihood on the validation set was used as stopping point.

Table 1 shows Gaussian Parzen window log-likelihood estimate for the MNIST dataset test data. 1000 samples were drawn from each 10 class and a Gaussian Parzen window was fitted to these samples. We then estimate the log-likelihood of the test set using the Parzen window distribution. (See [8] for more details of how this estimate is constructed.)

The conditional adversarial net results that we present are comparable with some other network based, but are outperformed by several other approaches – including non-conditional adversarial nets. We present these results more as a proof-of-concept than as demonstration of efficacy, and believe that with further exploration of hyper-parameter space and architecture that the conditional model should match or exceed the non-conditional results.

Fig 2 shows some of the generated samples. Each row is conditioned on one label and each column is a different generated sample.

Figure: Generated MNIST digits, each row conditioned on one label
Conditional DCGAN
My image generation results on CIFAR-100

Figure: (Left) Each row has the same label $y$ and each column has the same noisy vector $z$ (Middle) Set noisy vector as zero (Right) Set label vector as zero
Text feature extractor is trained by:

\[
\min \frac{1}{N} \sum_{n=1}^{N} \Delta(y_n, f_v(v_n)) + \Delta(y_n, f_t(t_n))
\]  

(3)

\[
f_v(v) = \arg\max_y \mathbb{E}_{t \sim \tau(y)} [\phi^T(v)\psi(t)]
\]  

(4)

\[
f_t(t) = \arg\max_y \mathbb{E}_{v \sim \nu(y)} [\phi^T(v)\psi(t)]
\]  

(5)

where \( \phi \) and \( \psi \) are image and text encoder, respectively. \( \tau(y) \) is the set of caption of class \( y \) and likewise \( \nu(y) \).
Generative Adversarial Text to Image Synthesis

Algorithm 1 GAN-CLS training algorithm with step size \( \alpha \), using minibatch SGD for simplicity.

1: **Input:** minibatch images \( x \), matching text \( t \), mis-matching \( \hat{t} \), number of training batch steps \( S \)
2: **for** \( n = 1 \) **to** \( S \) **do**
3: \( h \leftarrow \varphi(t) \) \{Encode matching text description\}
4: \( \hat{h} \leftarrow \varphi(\hat{t}) \) \{Encode mis-matching text description\}
5: \( z \sim \mathcal{N}(0,1)^Z \) \{Draw sample of random noise\}
6: \( \hat{x} \leftarrow G(z, h) \) \{Forward through generator\}
7: \( s_r \leftarrow D(x, h) \) \{real image, right text\}
8: \( s_w \leftarrow D(x, h) \) \{real image, wrong text\}
9: \( s_f \leftarrow D(\hat{x}, h) \) \{fake image, right text\}
10: \( \mathcal{L}_D \leftarrow \log(s_r) + (\log(1 - s_w) + \log(1 - s_f))/2 \)
11: \( D \leftarrow D - \alpha \partial \mathcal{L}_D / \partial D \) \{Update discriminator\}
12: \( \mathcal{L}_G \leftarrow \log(s_f) \)
13: \( G \leftarrow G - \alpha \partial \mathcal{L}_G / \partial G \) \{Update generator\}
14: **end for**

- Manifold interpolation for text:

\[
\min \mathbb{E}_{t_1, t_2 \sim p_{data}} [\log(1 - D(G(z, \beta t_1 + (1 - \beta t_2))))]
\]  

where \( t_1 \) and \( t_2 \) are samples drawn from text embedding.
Generative Adversarial Text to Image Synthesis

Figure: Generated bird images

an all black bird with a distinct thick, rounded bill.
this small bird has a yellow breast, brown crown, and black superciliary
tiny bird, with a tiny beak, tarsus and feet, a blue crown, blue coverts, and black cheek patch
this bird is different shades of brown all over with white and black spots on its head and back
the gray bird has a light grey head and grey webbed feet

a tiny bird, with a tiny beak, tarsus and feet, a blue crown, blue coverts, and black cheek patch

this small bird has a yellow breast, brown crown, and black superciliary
tiny bird, with a tiny beak, tarsus and feet, a blue crown, blue coverts, and black cheek patch
this bird is different shades of brown all over with white and black spots on its head and back
the gray bird has a light grey head and grey webbed feet

Figure: Generated flower images using GAN, GAN-CLS, GAN-INT and GAN-INT-CLS. All variants generated plausible images. Although some shapes of test categories were not seen during training (e.g. columns 3 and 4), the color information is preserved.

Results on CUB can be seen in Figure 3. We observe that images generated by GAN and GAN-CLS change depending on the text. They may get some color information right, but the images do not look real. However, GAN-INT and GAN-INT-CLS show plausible images that usually match all or at least part of the caption. We include additional analysis on the robustness of each GAN variant on the CUB dataset in the supplement.

Results on Oxford-102 can be seen in Figure 4. In this case, all four methods can generate plausible flower images that match the description. The basic GAN tends to have the most variety in flower morphology (i.e. one can see very different petal types if this part is left unspecified by the caption), while other methods tend to generate more class-consistent images. We speculate that it is easier to generate flowers, perhaps because birds have stronger structural regularities across species that make it easier for D to spot a fake bird than to spot a fake flower.

Many additional results with GAN-INT and GAN-INT-CLS as well as GAN-E2E (our end-to-end GAN-INT-CLS without pre-training the text encoder \( \phi (t) \)) for both CUB and Oxford-102 can be found in the supplementary.

5.2. Disentangling style and content

In this section we investigate the extent to which our model can separate style and content. By content, we mean the visual attributes of the bird itself, such as shape, size and color of each body part. By style, we mean all of the other factors of variation in the image such as background color and the pose orientation of the bird.
this tiny bird has a yellow breast, brown crown, and black superciliary

GT

this small bird has a blue crown, blue coverts, and black cheek patch

GAN

these flowers have petals that start off white in color and end in a dark purple towards the tips.

GAN - CLS

bright droopy yellow petals with burgundy streaks, and a yellow stigma.

GAN - INT

a flower with long pink petals and raised orange stamen.

GAN - INT - CLS

the flower shown has a blue petals with a white pistil in the center

Figure: Generated flower images
Text descriptions (content) Images (style)

The bird has a yellow breast with grey features and a small beak.

This is a large white bird with black wings and a red head.

A small bird with a black head and wings and features grey wings.

This bird has a white breast, brown and white coloring on its head and wings, and a thin pointy beak.

A small bird with white base and black stripes throughout its belly, head, and feathers.

A small sized bird that has a cream belly and a short pointed bill.

This bird is completely red.

This bird is completely white.

This is a yellow bird. The wings are bright blue.

Figure: Interpolation between two sentences

Figure: Transfering style
Generative Adversarial Text to Image Synthesis

Figure: Generated images on COCO validation set
Decompose the input vector into two parts: $z \rightarrow [z, c_1, \ldots, c_L]$

- Incompressible noise: $z$
- Latent code: $c = [c_1, \ldots, c_L]$

Maximize the mutual information $I(c; G(z, c))$

\[
I(c; G(z, c)) = H(c) - H(c|G(z, c))
\]

\[
= \mathbb{E}_{x \sim G(z,c)} \mathbb{E}_{c' \sim p(c|x)} [\log p(c'|x)] + H(c)
\]

\[
\geq \mathbb{E}_{x \sim G(z,c)} \mathbb{E}_{c' \sim p(c|x)} [\log Q(c'|x)] + H(c)
\]

\[
= \mathbb{E}_{x \sim G(z,c)} [\log Q(c|x)] + H(c)
\]

\[
= L_1(G, Q)
\]

Here $x \sim G(z, c)$ means $c \sim P_c(c), z \sim P_z(z), x = G(z, c)$. 

\[
I(c; G(z, c)) = \frac{1}{2} \left( \| \mu^x \|_2^2 + \| \mu^c \|_2^2 - \log \det \Sigma_x \right)
\]
InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets

- Rewrite \( z \sim P_{z}(z), x = G(z, c) \) as \( x \sim p(x|c) \)

\[
L_{1}(G, Q) = \mathbb{E}_{x \sim G(z, c)}[\log Q(c|x)] + H(c) \quad (12)
\]

\[
= \mathbb{E}_{c \sim P_{c}(c), x \sim p(x|c)}[\log Q(c|x)] + H(c) \quad (13)
\]

- Relationship to variational lower bound:

\[
\mathbb{E}_{x \sim P_{data}, c \sim q(c|x)}[\log p(x|c) + \log p(c)] \quad (14)
\]

- Objective function:

\[
\min_{G} \max_{D} \mathbb{E}_{x \sim P_{data}}[\log D(x)] + \mathbb{E}_{z \sim p_{z}(z), c \sim P_{c}(c)}[\log(1 - D(G(z, c)))] - \lambda \mathbb{E}_{c \sim P_{c}(c), x \sim p(x|c)}[\log Q(c|x)] \quad (15)
\]
InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets

- $c = [c_1, c_2, c_3]: c_1 \sim \text{Cat}(10), c_2, c_3 \sim \text{Unif}(-1, 1)$

(a) Varying $c_1$ on InfoGAN (Digit type)  
(b) Varying $c_1$ on regular GAN (No clear meaning)

(c) Varying $c_2$ from $-2$ to $2$ on InfoGAN (Rotation)  
(d) Varying $c_3$ from $-2$ to $2$ on InfoGAN (Width)

On the chairs dataset, DC-IGN can learn a continuous code that represents rotation. InfoGAN again is able to learn the same concept as a continuous code (Figure 4a) and we show in addition that InfoGAN is also able to continuously interpolate between similar chair types of different widths using a single continuous code (Figure 4b). In this experiment, we choose to model the latent factors with four categorical codes, $c_1, c_2, c_3, c_4 \sim \text{Cat}(K = 20, p = 0.05)$ and one continuous code $c_5 \sim \text{Unif}(-1, 1)$.

Next we evaluate InfoGAN on the Street View House Number (SVHN) dataset, which is significantly more challenging to learn an interpretable representation because it is noisy, containing images of variable-resolution and distracting digits, and it does not have multiple variations of the same object. In this experiment, we make use of four $10$-dimensional categorical variables and two uniform continuous variables as latent codes. We show two of the learned latent factors in Figure 5.

Finally we show in Figure 6 that InfoGAN is able to learn many visual concepts on another challenging dataset: CelebA [33], which includes $200,000$ celebrity images with large pose variations and background clutter. In this dataset, we model the latent variation as $10$ uniform categorical variables, each of dimension $10$. Surprisingly, even in this complicated dataset, InfoGAN can recover azimuth as in 3D images even though in this dataset no single face appears in multiple pose positions. Moreover InfoGAN can disentangle other highly semantic variations like presence or absence of glasses, hairstyles and emotion, demonstrating a level of visual understanding is acquired without any supervision.
\[ c = [c_1, \ldots, c_5] \colon c_i \sim \text{Unif}(-1, 1) \]

(a) Azimuth (pose)  
(b) Elevation  
(c) Lighting  
(d) Wide or Narrow
GAN vs VAE

- **Prior**
  - VAE
  - GAN
  - VAE/GAN

- **Image**
  - Posterior
  - Training data
  - Sample from prior

- **Sample from prior**
GAN vs VAE

- The latent code $z = [z_1, z_2]$: $z_i \sim \mathcal{N}(0, 1)$.
- $z_i$ varies from -1.5 to 1.5.

Figure: Generated digits trained with VAE (left) GAN (right) from Chunyuan.
Autoencoding beyond pixels using a learned similarity metric

- **VAE:** $z \sim Enc(x) = q(z|x), \hat{x} \sim Dec(z) = p(x|z)$:

  \[
  \mathcal{L}_{VAE} = -\mathbb{E}_{q(z|x)}[\log p(x|z)] + D_{KL}(q(z|x)\|p(z))
  \]

  \[
  = \mathcal{L}_{\text{pixel}}^{\text{like}} + \mathcal{L}_{\text{prior}}
  \]

- **GAN:** $\mathcal{L}_{GAN} = \log(\text{Dis}(x)) + \log(1 - \text{Dis}(\text{Gen}(z)))$

- A new reconstruction error:

  \[
  p(\text{Dis}_l(x)|z) = \mathcal{N}(\text{Dis}_l(x)|\text{Dis}_l(\hat{x}), I)
  \]

  \[
  \mathcal{L}_{\text{like}}^{\text{Dis}_l(x)} = -\mathbb{E}_{q(z|x)}[\log p(\text{Dis}_l(x)|z)]
  \]

- **Objective function:**

  \[
  \mathcal{L} = \mathcal{L}_{\text{prior}} + \mathcal{L}_{\text{like}}^{\text{Dis}_l(x)} + \mathcal{L}_{GAN}
  \]
Adversarial Autoencoder (AAE)

\[ \mathcal{L}_{AAE} = \mathcal{L}_{\text{pixel}}^{\text{like}} + \mathcal{L}_{\text{VAE}} \]

\[ = -\mathbb{E}_{q(z|x)}[\log p(x|z)] + \lambda \left\{ \mathbb{E}_{z \sim p(z)}[\log D(z)] + \mathbb{E}_{z \sim q(z)}[\log D(1 - G(z))] \right\} \]

(21)

where \( q(z) = \int q(z|x)p_{\text{data}}(x)dx \)
Adversarial Autoencoder (AAE)

Figure 2: Comparison of adversarial and variational autoencoder on MNIST. The hidden code \( z \) 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 \( z \) in the 2-D Gaussian adversarial
autoencoder.
An important difference between VAEs and adversarial autoencoders is that in VAEs, in order to back-propagate through the one-hot vector to the discriminative network, the one-hot vector has an extra label for training points with unknown classes.

Figure 3: Regularizing the hidden code by providing a one-hot vector to the discriminative network. A video showing the learnt TFD manifold can be found at http://www.comm.utoronto.ca/~makhzani/adv_ae/tfd.gif.

Figure 4: Leveraging label information to better regularize the hidden code. Top Row: Training the coding space. (a) Coding space of 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. (b) Samples generated by walking along the main swiss roll axis.

2.2 Relationship to GANs and GMMNs

Conditional Adversarial Autoencoder

Figure: Adversarial Autoencoder conditioned on labels: Model architecture (Left); Posterior samples from a mixture of 10 2-D Gaussian (Right top) and a swiss roll distribution (Right bottom).
Figure: Model architecture for supervised AAE (left) and semi-supervised AAE (right)
Table 2: Semi-supervised classification performance (error-rate) on MNIST and SVHN.

<table>
<thead>
<tr>
<th>Method</th>
<th>MNIST (100)</th>
<th>MNIST (1000)</th>
<th>MNIST (All)</th>
<th>SVHN (1000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NN Baseline</td>
<td>25.80</td>
<td>8.73</td>
<td>1.25</td>
<td>47.50</td>
</tr>
<tr>
<td>VAE (M1) + TSVM</td>
<td>11.82 (±0.25)</td>
<td>4.24 (±0.07)</td>
<td>-</td>
<td>55.33 (±0.11)</td>
</tr>
<tr>
<td>VAE (M2)</td>
<td>11.97 (±1.71)</td>
<td>3.60 (±0.56)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>VAE (M1 + M2)</td>
<td>3.33 (±0.14)</td>
<td>2.40 (±0.02)</td>
<td>0.96</td>
<td>36.02 (±0.10)</td>
</tr>
<tr>
<td>VAT</td>
<td>2.33</td>
<td>1.36</td>
<td>0.64 (±0.04)</td>
<td>24.63</td>
</tr>
<tr>
<td>CatGAN</td>
<td>1.91 (±0.1)</td>
<td>1.73 (±0.18)</td>
<td>0.91</td>
<td>-</td>
</tr>
<tr>
<td>Ladder Networks</td>
<td>1.06 (±0.37)</td>
<td>0.84 (±0.08)</td>
<td>0.57 (±0.02)</td>
<td>-</td>
</tr>
<tr>
<td>ADGM</td>
<td>0.96 (±0.02)</td>
<td>-</td>
<td>-</td>
<td>16.61 (±0.24)</td>
</tr>
<tr>
<td><strong>Adversarial Autoencoders</strong></td>
<td><strong>1.90 (±0.10)</strong></td>
<td><strong>1.60 (±0.08)</strong></td>
<td><strong>0.85 (±0.02)</strong></td>
<td><strong>17.70 (±0.30)</strong></td>
</tr>
</tbody>
</table>

Table 2: Semi-supervised classification performance (error-rate) on MNIST and SVHN.
Semi-supervised Adversarial Autoencoder

Figure: Unsupervised clustering of MNIST using the AAE with 16 clusters. Each row corresponds to one cluster.

<table>
<thead>
<tr>
<th>Method</th>
<th>MNIST (Unsupervised)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CatGAN [Springenberg, 2015] (20 clusters)</td>
<td>9.70</td>
</tr>
<tr>
<td>Adversarial Autoencoder (16 clusters)</td>
<td>9.55 (±2.05)</td>
</tr>
<tr>
<td>Adversarial Autoencoder (30 clusters)</td>
<td>4.10 (±1.13)</td>
</tr>
</tbody>
</table>

Table 3: Unsupervised clustering performance (error-rate) of the AAE on MNIST.
Figure: Dimensionality reduction with adversarial autoencoders: The representation is constructed by first mapping the one-hot label representation to an n dimensional cluster head representation and then adding the result to an n dimensional style representation.
Conclusion

GAN, DCGAN:  \[ \mathcal{L}_{GAN} = \mathbb{E}_{x \sim P_{data}} (\log D(x)) + \mathbb{E}_{z \sim p(z)} (\log D(1 - G(z))) \]

ConGAN, GAN-T2I:  \[ \mathcal{L} = \mathbb{E}_{x \sim P_{data}} (\log D(x|y)) + \mathbb{E}_{z \sim p(z)} (\log(1 - D(G(z|y)|y))) \]

Info GAN:  \[ \mathcal{L} = \mathcal{L}_{GAN} + \text{Latent code reconstruction} \]

VAE/GAN:  \[ \mathcal{L} = \mathcal{L}_{GAN} + \text{Representation reconstruction} \]

AAE:  \[ \mathcal{L} = \text{Reconstruction} + \mathcal{L}_{GAN} \]