Advanced topics in deep generative models

Jakob Verbeek & Thomas Lucas
INRIA, Grenoble, France

Breaking the Surface 2019
Biograd na Moru, Croatia

Part I

Improving Variational Auto-encoders

Improving variational autoencoders

Improving variational autoencoders

- **Reminder:** VAEs optimize the ELBO, with a KL divergence to bound the data log-likelihood

$$F(\mathbf{x}, \theta, \phi) = \ln p(\mathbf{x}) - D\left(q_\phi(\mathbf{z}|\mathbf{x}) || p(\mathbf{z}|\mathbf{x})\right) \tag{1}$$

Improving variational autoencoders

- **Reminder:** VAEs optimize the ELBO, with a KL divergence to bound the data log-likelihood

$$F(\mathbf{x}, \theta, \phi) = \ln p(\mathbf{x}) - D\left(q_\phi(\mathbf{z}|\mathbf{x}) || p(\mathbf{z}|\mathbf{x})\right) \tag{1}$$

  - Generally true posterior is not Gaussian: loose bound

Improving variational autoencoders

- **Reminder:** VAEs optimize the ELBO, with a KL divergence to bound the data log-likelihood

$$F(\mathbf{x}, \theta, \phi) = \ln p(\mathbf{x}) - D\left(q_\phi(\mathbf{z}|\mathbf{x}) || p(\mathbf{z}|\mathbf{x})\right) \tag{1}$$

  - Generally true posterior is not Gaussian: loose bound
  - Encourages true posterior to match variational factored Gaussian produced by recognition net

Improving variational autoencoders

- **Reminder:** VAEs optimize the ELBO, with a KL divergence to bound the data log-likelihood

$$F(\mathbf{x}, \theta, \phi) = \ln p(\mathbf{x}) - D\left(q_\phi(\mathbf{z}|\mathbf{x}) || p(\mathbf{z}|\mathbf{x})\right) \tag{1}$$

  - Generally true posterior is not Gaussian: loose bound
  - Encourages true posterior to match variational factored Gaussian produced by recognition net

- Making progress

Improving variational autoencoders

- **Reminder:** VAEs optimize the ELBO, with a KL divergence to bound the data log-likelihood

$$F(\mathbf{x}, \theta, \phi) = \ln p(\mathbf{x}) - D\left(q_\phi(\mathbf{z}|\mathbf{x}) || p(\mathbf{z}|\mathbf{x})\right) \quad (1)$$

  - Generally true posterior is not Gaussian: loose bound
  - Encourages true posterior to match variational factored Gaussian produced by recognition net

- Making progress
  1. More accurate bound for given posterior

Improving variational autoencoders

- **Reminder:** VAEs optimize the ELBO, with a KL divergence to bound the data log-likelihood

$$F(\mathbf{x}, \theta, \phi) = \ln p(\mathbf{x}) - D\left(q_\phi(\mathbf{z}|\mathbf{x}) || p(\mathbf{z}|\mathbf{x})\right) \tag{1}$$

  - Generally true posterior is not Gaussian: loose bound
  - Encourages true posterior to match variational factored Gaussian produced by recognition net

- Making progress
  1. More accurate bound for given posterior
  2. Enlarge the family of variational posteriors
     - Hierarchical latent variables
     - Improved flexibility with flows

Importance weighted autoencoders [Burda et al., 2016]

- **Construct tighter lower bound using importance sampling**

Importance weighted autoencoders [Burda et al., 2016]

- **Construct tighter lower bound using importance sampling**
- Define importance weights $w(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}, \mathbf{z})/q_\phi(\mathbf{z}|\mathbf{x})$

Importance weighted autoencoders [Burda et al., 2016]

- **Construct tighter lower bound using importance sampling**
- Define importance weights $w(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}, \mathbf{z}) / q_\phi(\mathbf{z}|\mathbf{x})$

$$F_k(\mathbf{x}, \theta, \phi) = \mathbb{E}_{\mathbf{z}_1, \ldots, \mathbf{z}_k \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\ln \frac{1}{k} \sum_{i=1}^{k} w(\mathbf{x}, \mathbf{z}_i) \right]$$

Importance weighted autoencoders [Burda et al., 2016]

- **Construct tighter lower bound using importance sampling**
- Define importance weights $w(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}, \mathbf{z})/q_\phi(\mathbf{z}|\mathbf{x})$

$$
\begin{aligned}
F_k(\mathbf{x}, \theta, \phi) &= \mathbb{E}_{\mathbf{z}_1, \ldots, \mathbf{z}_k \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\ln \frac{1}{k} \sum_{i=1}^{k} w(\mathbf{x}, \mathbf{z}_i) \right] \\
&\leq \ln \mathbb{E}_{\mathbf{z}_1, \ldots, \mathbf{z}_k \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\frac{1}{k} \sum_{i=1}^{k} w(\mathbf{x}, \mathbf{z}_i) \right]
\end{aligned}
$$

Importance weighted autoencoders [Burda et al., 2016]

- **Construct tighter lower bound using importance sampling**
- Define importance weights $w(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}, \mathbf{z}) / q_\phi(\mathbf{z}|\mathbf{x})$

$$
\begin{aligned}
F_k(\mathbf{x}, \theta, \phi) &= \mathbb{E}_{\mathbf{z}_1, \ldots, \mathbf{z}_k \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\ln \frac{1}{k} \sum_{i=1}^{k} w(\mathbf{x}, \mathbf{z}_i) \right] \\
&\leq \ln \mathbb{E}_{\mathbf{z}_1, \ldots, \mathbf{z}_k \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\frac{1}{k} \sum_{i=1}^{k} w(\mathbf{x}, \mathbf{z}_i) \right] \\
&= \ln \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [w(\mathbf{x}, \mathbf{z})]
\end{aligned}
$$

Importance weighted autoencoders [Burda et al., 2016]

- **Construct tighter lower bound using importance sampling**
- Define importance weights $w(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}, \mathbf{z}) / q_\phi(\mathbf{z}|\mathbf{x})$

$$
\begin{aligned}
F_k(\mathbf{x}, \theta, \phi) &= \mathbb{E}_{\mathbf{z}_1, \ldots, \mathbf{z}_k \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\ln \frac{1}{k} \sum_{i=1}^{k} w(\mathbf{x}, \mathbf{z}_i) \right] \\
&\leq \ln \mathbb{E}_{\mathbf{z}_1, \ldots, \mathbf{z}_k \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\frac{1}{k} \sum_{i=1}^{k} w(\mathbf{x}, \mathbf{z}_i) \right] \\
&= \ln \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [w(\mathbf{x}, \mathbf{z})] \\
&= \ln p(\mathbf{x})
\end{aligned}
$$

Importance weighted autoencoders [Burda et al., 2016]

- **Construct tighter lower bound using importance sampling**
- Define importance weights $w(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}, \mathbf{z}) / q_\phi(\mathbf{z}|\mathbf{x})$

$$
\begin{aligned}
F_k(\mathbf{x}, \theta, \phi) &= \mathbb{E}_{\mathbf{z}_1, \ldots, \mathbf{z}_k \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\ln \frac{1}{k} \sum_{i=1}^{k} w(\mathbf{x}, \mathbf{z}_i) \right] \\
&\leq \ln \mathbb{E}_{\mathbf{z}_1, \ldots, \mathbf{z}_k \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\frac{1}{k} \sum_{i=1}^{k} w(\mathbf{x}, \mathbf{z}_i) \right] \\
&= \ln \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [w(\mathbf{x}, \mathbf{z})] \\
&= \ln p(\mathbf{x})
\end{aligned}
$$

1. VAE lower bound recovered for $k = 1$

Importance weighted autoencoders [Burda et al., 2016]

- **Construct tighter lower bound using importance sampling**
- Define importance weights $w(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}, \mathbf{z})/q_\phi(\mathbf{z}|\mathbf{x})$

$$
\begin{aligned}
F_k(\mathbf{x}, \theta, \phi) &= \mathbb{E}_{\mathbf{z}_1,\ldots,\mathbf{z}_k \sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\ln \frac{1}{k}\sum_{i=1}^{k} w(\mathbf{x}, \mathbf{z}_i)\right] \\
&\leq \ln \mathbb{E}_{\mathbf{z}_1,\ldots,\mathbf{z}_k \sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\frac{1}{k}\sum_{i=1}^{k} w(\mathbf{x}, \mathbf{z}_i)\right] \\
&= \ln \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})}[w(\mathbf{x}, \mathbf{z})] \\
&= \ln p(\mathbf{x})
\end{aligned}
$$

1. VAE lower bound recovered for $k = 1$
2. More samples tighten the bound: $F_k \leq F_{k+1} \leq \ln p(\mathbf{x})$

Importance weighted autoencoders [Burda et al., 2016]

- **Construct tighter lower bound using importance sampling**
- Define importance weights $w(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}, \mathbf{z}) / q_\phi(\mathbf{z}|\mathbf{x})$

$$
\begin{aligned}
F_k(\mathbf{x}, \theta, \phi) &= \mathbb{E}_{\mathbf{z}_1, \ldots, \mathbf{z}_k \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\ln \frac{1}{k} \sum_{i=1}^{k} w(\mathbf{x}, \mathbf{z}_i) \right] \\
&\leq \ln \mathbb{E}_{\mathbf{z}_1, \ldots, \mathbf{z}_k \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\frac{1}{k} \sum_{i=1}^{k} w(\mathbf{x}, \mathbf{z}_i) \right] \\
&= \ln \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [w(\mathbf{x}, \mathbf{z})] \\
&= \ln p(\mathbf{x})
\end{aligned}
$$

1. VAE lower bound recovered for $k = 1$
2. More samples tighten the bound: $F_k \leq F_{k+1} \leq \ln p(\mathbf{x})$
3. If the weights are bounded, then $F_k \to \ln p(\mathbf{x})$ as $k \to \infty$

Importance weighted autoencoders [Burda et al., 2016]

- **Construct tighter lower bound using importance sampling**
- Define importance weights $w(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}, \mathbf{z}) / q_\phi(\mathbf{z}|\mathbf{x})$

$$
\begin{aligned}
F_k(\mathbf{x}, \theta, \phi) &= \mathbb{E}_{\mathbf{z}_1, \dots, \mathbf{z}_k \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\ln \frac{1}{k} \sum_{i=1}^{k} w(\mathbf{x}, \mathbf{z}_i) \right] \\
&\leq \ln \mathbb{E}_{\mathbf{z}_1, \dots, \mathbf{z}_k \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\frac{1}{k} \sum_{i=1}^{k} w(\mathbf{x}, \mathbf{z}_i) \right] \\
&= \ln \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [w(\mathbf{x}, \mathbf{z})] \\
&= \ln p(\mathbf{x})
\end{aligned}
$$

1. VAE lower bound recovered for $k = 1$
2. More samples tighten the bound: $F_k \leq F_{k+1} \leq \ln p(\mathbf{x})$
3. If the weights are bounded, then $F_k \to \ln p(\mathbf{x})$ as $k \to \infty$

- Use as objective to train models,**e.g**. using $k \approx 10$

Importance weighted autoencoders [Burda et al., 2016]

- **Construct tighter lower bound using importance sampling**
- Define importance weights $w(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}, \mathbf{z}) / q_\phi(\mathbf{z}|\mathbf{x})$

$$
\begin{aligned}
F_k(\mathbf{x}, \theta, \phi) &= \mathbb{E}_{\mathbf{z}_1, \ldots, \mathbf{z}_k \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\ln \frac{1}{k} \sum_{i=1}^{k} w(\mathbf{x}, \mathbf{z}_i) \right] \\
&\leq \ln \mathbb{E}_{\mathbf{z}_1, \ldots, \mathbf{z}_k \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\frac{1}{k} \sum_{i=1}^{k} w(\mathbf{x}, \mathbf{z}_i) \right] \\
&= \ln \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [w(\mathbf{x}, \mathbf{z})] \\
&= \ln p(\mathbf{x})
\end{aligned}
$$

1. VAE lower bound recovered for $k = 1$
2. More samples tighten the bound: $F_k \leq F_{k+1} \leq \ln p(\mathbf{x})$
3. If the weights are bounded, then $F_k \to \ln p(\mathbf{x})$ as $k \to \infty$

- Use as objective to train models,**e.g**. using $k \approx 10$
- Use as likelihood estimator, **e.g**. with $k \approx 10^3$

Training procedure importance weighted autoencoders

- Gradients of importance weighted lower bound

$$\nabla F_k(\mathbf{x}) = \mathbb{E}_{\mathbf{z}_{1:k} \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\sum_{i=1}^{k} \widetilde{w}_i \nabla \big(\ln p(\mathbf{x}, \mathbf{z}_i) - \ln q_\phi(\mathbf{z}_i|\mathbf{x}) \big) \right]$$

Training procedure importance weighted autoencoders

- Gradients of importance weighted lower bound

$$\nabla F_k(\mathbf{x}) = \mathbb{E}_{\mathbf{z}_{1:k} \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\sum_{i=1}^{k} \widetilde{w}_i \nabla \big(\ln p(\mathbf{x}, \mathbf{z}_i) - \ln q_\phi(\mathbf{z}_i|\mathbf{x}) \big) \right]$$

- Similar to VAE, but samples weighted w.r.t. true posterior

$$\widetilde{w}_i = p(\mathbf{z}_i|\mathbf{x}) / q_\phi(\mathbf{z}_i|\mathbf{x}) \sum_{j=1}^{k} w(\mathbf{x}, \mathbf{z}_j) \tag{2}$$

Training procedure importance weighted autoencoders

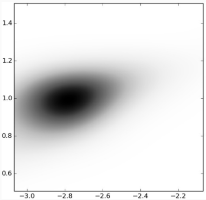
- Gradients of importance weighted lower bound

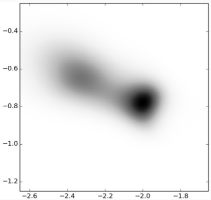
$$\nabla F_k(\mathbf{x}) = \mathbb{E}_{\mathbf{z}_{1:k} \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\sum_{i=1}^{k} \widetilde{w}_i \nabla \big(\ln p(\mathbf{x}, \mathbf{z}_i) - \ln q_\phi(\mathbf{z}_i|\mathbf{x}) \big) \right]$$

- Similar to VAE, but samples weighted w.r.t. true posterior

$$\widetilde{w}_i = p(\mathbf{z}_i|\mathbf{x}) / q_\phi(\mathbf{z}_i|\mathbf{x}) \sum_{j=1}^{k} w(\mathbf{x}, \mathbf{z}_j) \quad (2)$$

- Allows for more accurate models with complex posteriors

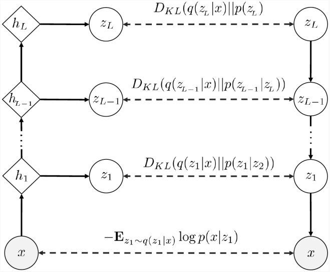




From [Burda et al., 2016]: True posterior $p(\mathbf{z}|\mathbf{x})$ VAE (left) and IW-VAE (right)

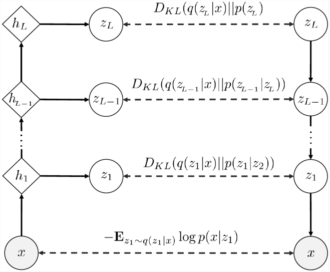
Top-down hierarchical sampling

- Multiple levels of latent variables at increasing resolutions



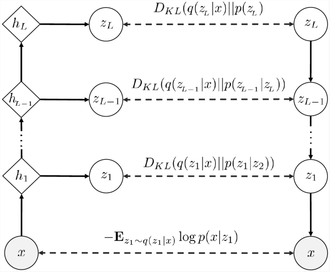
Top-down hierarchical sampling

- Multiple levels of latent variables at increasing resolutions
- Autoregressive distribution $p(z_1|z_2)$ over latent variables in 2D grid



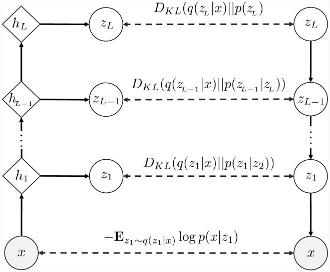
Top-down hierarchical sampling

- Multiple levels of latent variables at increasing resolutions
- Autoregressive distribution $p(z_1|z_2)$ over latent variables in 2D grid
- Sample latent variables in same order when encoding or sampling



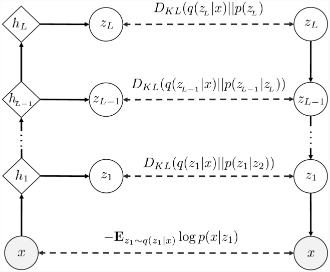
Top-down hierarchical sampling

- Multiple levels of latent variables at increasing resolutions
- Autoregressive distribution $p(z_1|z_2)$ over latent variables in 2D grid
- Sample latent variables in same order when encoding or sampling
- Posterior no longer Gaussian
  - $q(\mathbf{z}_1, \mathbf{z}_2|\mathbf{x}) = q(\mathbf{z}_1|\mathbf{x}, \mathbf{z}_2)q(\mathbf{z}_2|\mathbf{x})$



Top-down hierarchical sampling

- Multiple levels of latent variables at increasing resolutions
- Autoregressive distribution $p(z_1|z_2)$ over latent variables in 2D grid
- Sample latent variables in same order when encoding or sampling
- Posterior no longer Gaussian
  - $q(\mathbf{z}_1, \mathbf{z}_2|\mathbf{x}) = q(\mathbf{z}_1|\mathbf{x}, \mathbf{z}_2)q(\mathbf{z}_2|\mathbf{x})$
- Extended VAE log-likelihood bound



$$F = \ln p(\mathbf{x}) - D_{KL}(q(\mathbf{z}_{1:L}|\mathbf{x})||p(\mathbf{z}_{1:L}|\mathbf{x})$$
$$= \underbrace{\mathbb{E}_{q(\mathbf{z}_1|\mathbf{x})}[\ln p(\mathbf{x}|\mathbf{z}_1)]}_{\text{Reconstruction}} - \underbrace{\sum_{i=1}^{L} \mathbb{E}_{q(\mathbf{z}_{i+1})}[D_{KL}(q(\mathbf{z}_i|\mathbf{x})||p(\mathbf{z}_i|\mathbf{z}_{i+1})])}_{\text{Regularization}}$$

Variational inference with normalizing flows

- Variational inference (in VAE) uses limited class of posteriors
    - For example, Gaussian with diagonal covariance
    - Optimizing loose bound on data log-likelihood

Variational inference with normalizing flows

- Variational inference (in VAE) uses limited class of posteriors
  - For example, Gaussian with diagonal covariance
  - Optimizing loose bound on data log-likelihood
- Improve posterior approximation with invertible flow

Variational inference with normalizing flows

- Variational inference (in VAE) uses limited class of posteriors
  - For example, Gaussian with diagonal covariance
  - Optimizing loose bound on data log-likelihood

- Improve posterior approximation with invertible flow

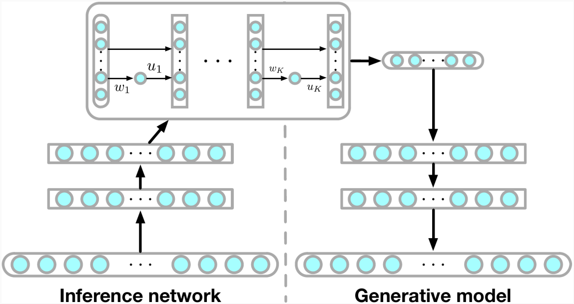


Figure from [Rezende and Mohamed, 2015]

Normalizing flows

- Let density "flow" through set of invertible transformations

$$
\begin{aligned}
\mathbf{z}_K &= f_K \circ \cdots \circ f_2 \circ f_1(\mathbf{z}_0), \\
\ln q_K(\mathbf{z}_K) &= \ln q_0(\mathbf{z}_0) - \sum_{k=1}^{K} \ln \left| \det \frac{\partial f_k}{\partial \mathbf{z}_k} \right|
\end{aligned}
$$

Normalizing flows

- Let density "flow" through set of invertible transformations

$$\mathbf{z}_K = f_K \circ \cdots \circ f_2 \circ f_1(\mathbf{z}_0),$$
$$\ln q_K(\mathbf{z}_K) = \ln q_0(\mathbf{z}_0) - \sum_{k=1}^{K} \ln \left| \det \frac{\partial f_k}{\partial \mathbf{z}_k} \right|$$

- $O(D)$ determinant, rather than $O(D^3)$, for planar and radial flows

$$f(\mathbf{z}) = \mathbf{z} + \mathbf{u} h(\mathbf{w}^\top \mathbf{z} + b)$$
$$f(\mathbf{z}) = \mathbf{z} + \beta h(\alpha, r)(\mathbf{z} - \mathbf{z_0})$$

Normalizing flows

- Let density "flow" through set of invertible transformations

$$
\begin{aligned}
\mathbf{z}_K &= f_K \circ \cdots \circ f_2 \circ f_1(\mathbf{z}_0), \\
\ln q_K(\mathbf{z}_K) &= \ln q_0(\mathbf{z}_0) - \sum_{k=1}^{K} \ln \left| \det \frac{\partial f_k}{\partial \mathbf{z}_k} \right|
\end{aligned}
$$

- $O(D)$ determinant, rather than $O(D^3)$, for planar and radial flows

$$
\begin{aligned}
f(\mathbf{z}) &= \mathbf{z} + \mathbf{u} h(\mathbf{w}^\top \mathbf{z} + b) \\
f(\mathbf{z}) &= \mathbf{z} + \beta h(\alpha, r)(\mathbf{z} - \mathbf{z_0})
\end{aligned}
$$

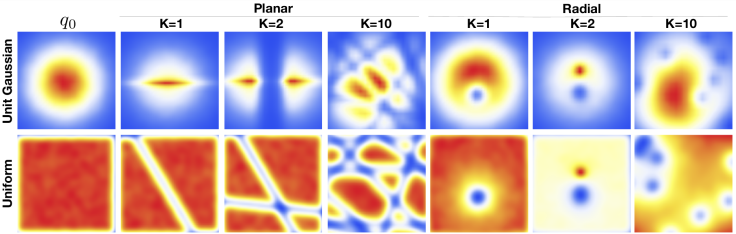
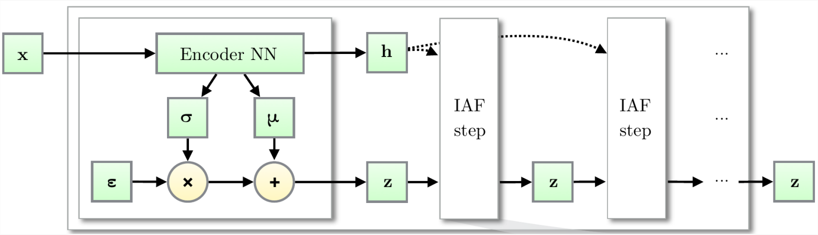


Figure from [Rezende and Mohamed, 2015]

Autoregressive flow [Kingma et al., 2016]

- Restictive flows in [Rezende and Mohamed, 2015]
  - Planar flow similar to MLP with single hidden unit
- Use autoregressive transformations in flow
  - Rich and tractable class of transformations
  - Fewer transformations needed



Autoregressive flow [Kingma et al., 2016]

- Class of affine transformations with respect to $\mathbf{z}$

$$\mathbf{z}^{t+1} = \mu^t + \sigma^t \odot \mathbf{z}^t$$

Autoregressive flow [Kingma et al., 2016]

- Class of affine transformations with respect to $\mathbf{z}$

$$\mathbf{z}^{t+1} = \mu^t + \sigma^t \odot \mathbf{z}^t$$

- Autoregressive computation of affine parameters

$$\mu_{i+1}^t = f(\mathbf{z}_{1:i}^t) \qquad \sigma_{i+1}^t = g(\mathbf{z}_{1:i}^t)$$

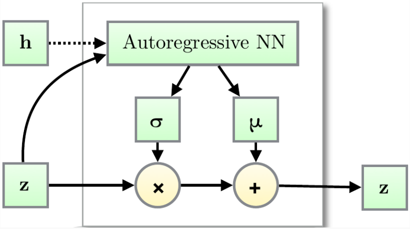
Autoregressive flow [Kingma et al., 2016]

- Class of affine transformations with respect to $\mathbf{z}$

$$\mathbf{z}^{t+1} = \mu^t + \sigma^t \odot \mathbf{z}^t$$

- Autoregressive computation of affine parameters

$$\mu^t_{i+1} = f(\mathbf{z}^t_{1:i}) \qquad \sigma^t_{i+1} = g(\mathbf{z}^t_{1:i})$$



Autoregressive flow [Kingma et al., 2016]

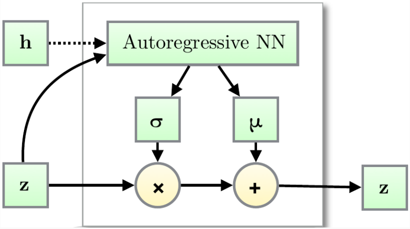
- Class of affine transformations with respect to $\mathbf{z}$

$$\mathbf{z}^{t+1} = \mu^t + \sigma^t \odot \mathbf{z}^t$$

- Autoregressive computation of affine parameters

$$\mu^t_{i+1} = f(\mathbf{z}^t_{1:i}) \qquad \sigma^t_{i+1} = g(\mathbf{z}^t_{1:i})$$

- Triangular Jacobian, log-determinant $\sum_{i=1}^{D} \log \sigma_i^t$



Autoregressive flow [Kingma et al., 2016]

- Class of affine transformations with respect to $\mathbf{z}$

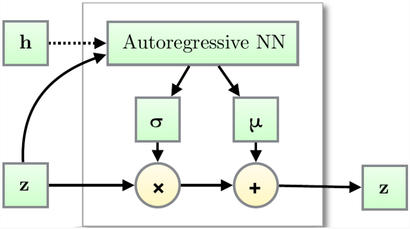
$$\mathbf{z}^{t+1} = \mu^t + \sigma^t \odot \mathbf{z}^t$$

- Autoregressive computation of affine parameters

$$\mu^t_{i+1} = f(\mathbf{z}^t_{1:i}) \qquad \sigma^t_{i+1} = g(\mathbf{z}^t_{1:i})$$

- Triangular Jacobian, log-determinant $\sum_{i=1}^{D} \log \sigma^t_i$

- Free to chose form of autoregressive NN dependency



Improved VAE — Recap

Ways to improve the tightness of the ELBO:

- Importance weighted autoencoder
- Hierarchical top-down sampling
- Density flow transformation

Beyond conditional independence assumption in VAE

- Standard VAE decoders assumes conditional independence

$$p(\mathbf{x}|\mathbf{z}) = \prod_{i=1}^{D} p(x_i|\mathbf{z}), \quad (3)$$

$$p(x_i|\mathbf{z}) = \mathcal{N}\left(x_i; f_\theta^\mu(\mathbf{z})_i, f_\theta^\sigma(\mathbf{z})_i\right) \quad (4)$$

Beyond conditional independence assumption in VAE

- Standard VAE decoders assumes conditional independence

$$p(\mathbf{x}|\mathbf{z}) = \prod_{i=1}^{D} p(x_i|\mathbf{z}), \tag{3}$$

$$p(x_i|\mathbf{z}) = \mathcal{N}\left(x_i; f_\theta^\mu(\mathbf{z})_i, f_\theta^\sigma(\mathbf{z})_i\right) \tag{4}$$

- Conditional log-likelihood is ℓ_2 reconstruction term

Beyond conditional independence assumption in VAE

- Standard VAE decoders assumes conditional independence

$$p(\mathbf{x}|\mathbf{z}) = \prod_{i=1}^{D} p(x_i|\mathbf{z}), \tag{3}$$

$$p(x_i|\mathbf{z}) = \mathcal{N}\left(x_i; f_\theta^\mu(\mathbf{z})_i, f_\theta^\sigma(\mathbf{z})_i\right) \tag{4}$$

- Conditional log-likelihood is ℓ_2 reconstruction term
- Bad metric of image similarity

Beyond conditional independence assumption in VAE

- Standard VAE decoders assumes conditional independence

$$p(\mathbf{x}|\mathbf{z}) = \prod_{i=1}^{D} p(x_i|\mathbf{z}), \tag{3}$$

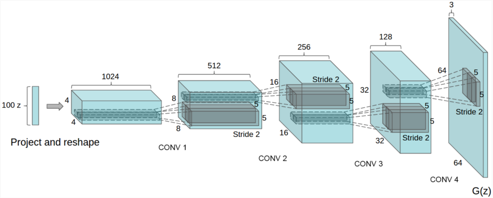
$$p(x_i|\mathbf{z}) = \mathcal{N}\left(x_i; f_\theta^\mu(\mathbf{z})_i, f_\theta^\sigma(\mathbf{z})_i\right) \tag{4}$$

- Conditional log-likelihood is ℓ_2 reconstruction term
- Bad metric of image similarity
- Leads to blurry images, and over-generalization

Hybrid PixelCNN-VAE model [Gulrajani et al., 2017b, Chen et al., 2017]

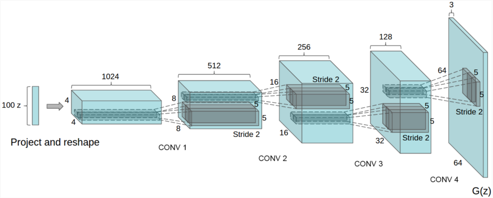
Hybrid PixelCNN-VAE model [Gulrajani et al., 2017b, Chen et al., 2017]

- Variational autoencoder
  - Latent variable **z** generates global dependencies
  - Pixels conditionally independent given code

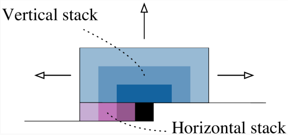


Hybrid PixelCNN-VAE model [Gulrajani et al., 2017b, Chen et al., 2017]

- Variational autoencoder
  - Latent variable **z** generates global dependencies
  - Pixels conditionally independent given code

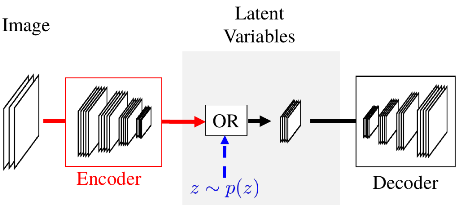


- Autoregressive PixelCNN
  - Needs many layers to induce long-range dependencies
  - Doesn't learn latent representation



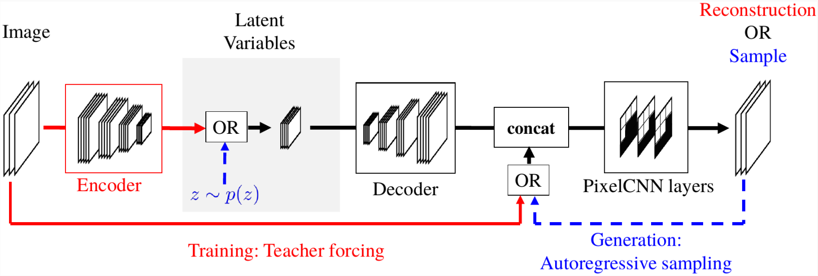
Hybrid PixelVAE model [Gulrajani et al., 2017b]

- Latent var. input to deterministic upsampling decoder $f(\mathbf{z})$



Hybrid PixelVAE model [Gulrajani et al., 2017b]

- Latent var. input to deterministic upsampling decoder $f(\mathbf{z})$
- Pixel-CNN layers induce local pixel dependencies

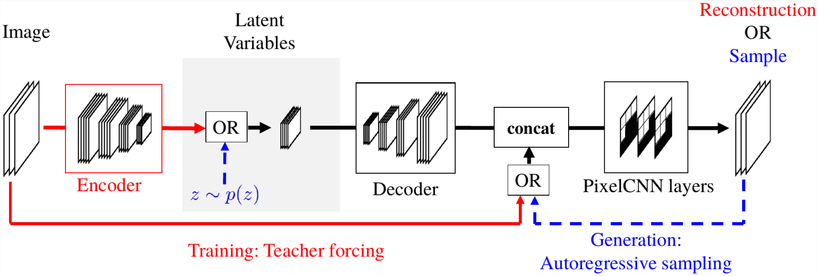


Hybrid PixelVAE model [Gulrajani et al., 2017b]

- Latent var. input to deterministic upsampling decoder $f(\mathbf{z})$
- Pixel-CNN layers induce local pixel dependencies

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; 0, I), \quad (5)$$

$$p(\mathbf{x}) = \int_{\mathbf{z}} p(\mathbf{z}) \prod_i p(x_i | \mathbf{x}_{<i}, f(\mathbf{z})) \quad (6)$$



Samples PixelVAE model LSUN dataset

- Model with three levels of stochasticity
  - Latent variables at 1×1
  - Latent variables at 8×8
  - PixelCNN at 64×64

Samples PixelVAE model LSUN dataset

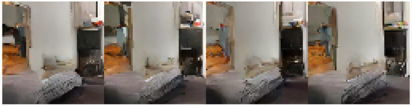
- Model with three levels of stochasticity
  - Latent variables at 1×1
  - Latent variables at 8×8
  - PixelCNN at 64×64

Re-sampling PixelCNN only

Samples PixelVAE model LSUN dataset

- Model with three levels of stochasticity
  - Latent variables at 1×1
  - Latent variables at 8×8
  - PixelCNN at 64×64

Re-sampling PixelCNN only



Samples PixelVAE model LSUN dataset

- Model with three levels of stochasticity
  - Latent variables at 1×1
  - Latent variables at 8×8
  - PixelCNN at 64×64

Re-sampling PixelCNN only

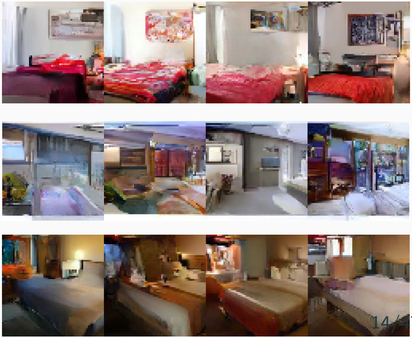
Samples PixelVAE model LSUN dataset

- Model with three levels of stochasticity
    - Latent variables at 1×1
    - Latent variables at 8×8
    - PixelCNN at 64×64

- Hierarchical representation learning

Re-sampling PixelCNN only

Re-sampling 8×8 + PixelCNN

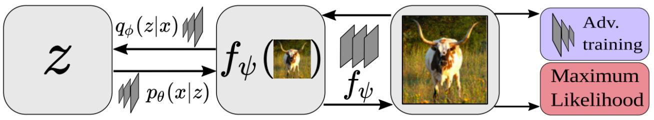


Hybrid VAE-Flow model [Lucas et al., 2019]

- Use flow-model to induce pixel dependencies and non-Gaussianity
- Avoid slow-sampling of pixelCNN, allows for adversarial training

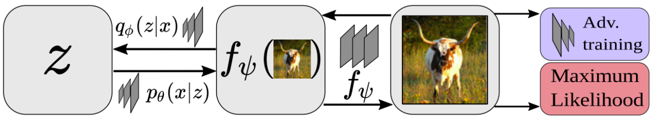
Hybrid VAE-Flow model [Lucas et al., 2019]

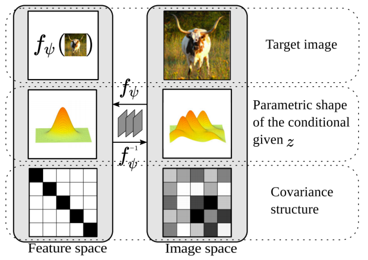
- Use flow-model to induce pixel dependencies and non-Gaussianity
- Avoid slow-sampling of pixelCNN, allows for adversarial training



Hybrid VAE-Flow model [Lucas et al., 2019]

- Use flow-model to induce pixel dependencies and non-Gaussianity
- Avoid slow-sampling of pixelCNN, allows for adversarial training



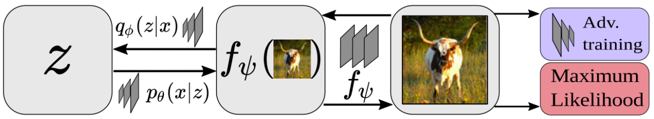


Hybrid VAE-Flow model [Lucas et al., 2019]

- Simple prior on latents, factored conditional on feature space

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; 0, I), \quad (7)$$

$$p_{\mathbf{y}}(\mathbf{y}|\mathbf{z}) = \mathcal{N}(\mathbf{y}; \mu(\mathbf{z}), \mathrm{diag}(\sigma(\mathbf{z}))) \quad (8)$$



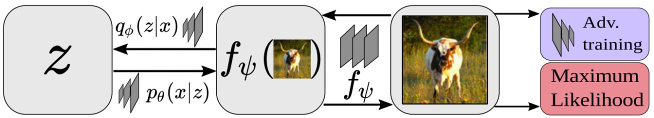
Hybrid VAE-Flow model [Lucas et al., 2019]

- Simple prior on latents, factored conditional on feature space

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; 0, I), \quad (7)$$

$$p_{\mathbf{y}}(\mathbf{y}|\mathbf{z}) = \mathcal{N}(\mathbf{y}; \mu(\mathbf{z}), \mathrm{diag}(\sigma(\mathbf{z}))) \quad (8)$$

- Flow across feature space and image space: $\mathbf{x} = f^{-1}(\mathbf{y})$



Hybrid VAE-Flow model [Lucas et al., 2019]

- Simple prior on latents, factored conditional on feature space

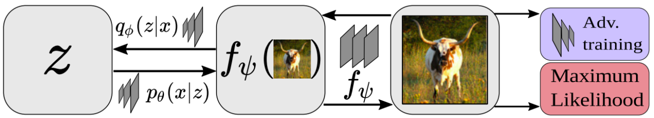
$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; 0, I), \tag{7}$$

$$p_{\mathbf{y}}(\mathbf{y}|\mathbf{z}) = \mathcal{N}(\mathbf{y}; \mu(\mathbf{z}), \mathrm{diag}(\sigma(\mathbf{z}))) \tag{8}$$

- Flow across feature space and image space: $\mathbf{x} = f^{-1}(\mathbf{y})$

- Variational inference network on latent space given image

$$q(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{y}; m(\mathbf{x}), \mathrm{diag}(s(\mathbf{x}))) \tag{9}$$



Hybrid VAE-Flow model [Lucas et al., 2019]

- Simple prior on latents, factored conditional on feature space

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; 0, I), \quad (7)$$

$$p_{\mathbf{y}}(\mathbf{y}|\mathbf{z}) = \mathcal{N}(\mathbf{y}; \mu(\mathbf{z}), \mathrm{diag}(\sigma(\mathbf{z}))) \quad (8)$$

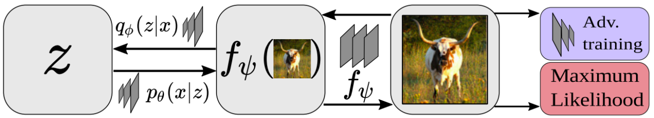
- Flow across feature space and image space: $\mathbf{x} = f^{-1}(\mathbf{y})$

- Variational inference network on latent space given image

$$q(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{y}; m(\mathbf{x}), \mathrm{diag}(s(\mathbf{x}))) \quad (9)$$

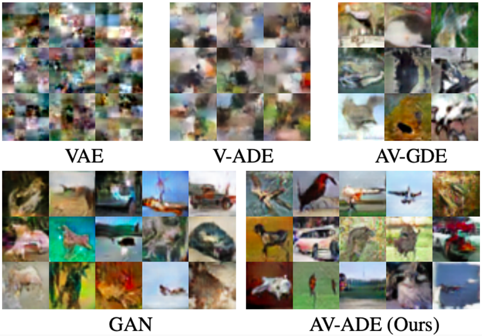
- Evidence lower-bound with change of variables

$$\ln p(\mathbf{x}) \geq \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\ln p_{\mathbf{y}}(f(\mathbf{x})|\mathbf{z})] - D_{\mathrm{KL}}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{y})) + \ln \left| \det \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \right|$$



Hybrid VAE-Flow model - Ablation

- Adversarial training critical for good sample quality
- MLE critical for good held-out likelihoods
- Flow improves both likelihoods and sample quality



	f_ψ	Adv.	MLE	BPD ↓	IS ↑	FID ↓
GAN	×	✓	×	[7.0]	6.8	31.4
VAE	×	×	✓	4.4	2.0	171.0
V-ADE†	✓	×	✓	3.5	3.0	112.0
AV-GDE	×	✓	✓	4.4	5.1	58.6
AV-ADE†	✓	✓	✓	3.9	7.1	28.0

Table 1: Quantitative results. † : Parameter count decreased by 1.4% to compensate for f_ψ. [Square brackets] denote that the value is approximated, see Section 5.

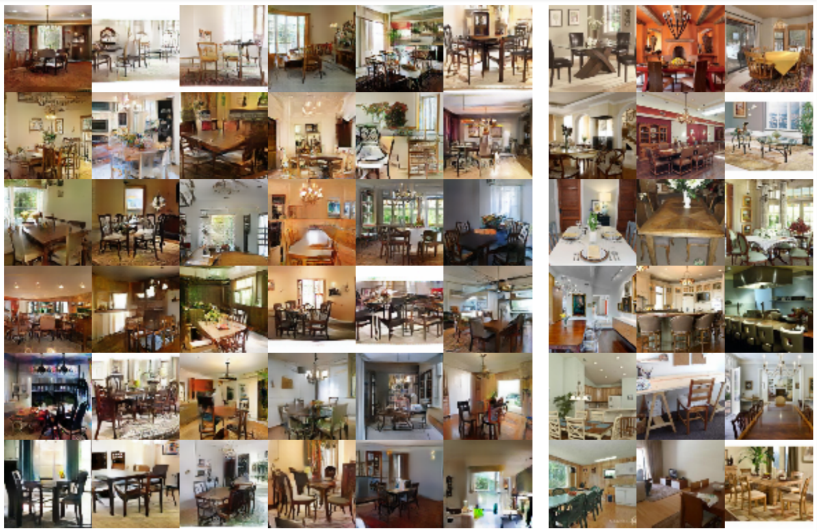
Figure 5: Samples from GAN and VAE baselines, our V-ADE, AV-GDE and AV-ADE models, all trained on CIFAR-10.

Hybrid VAE-Flow model - Comparison to Glow

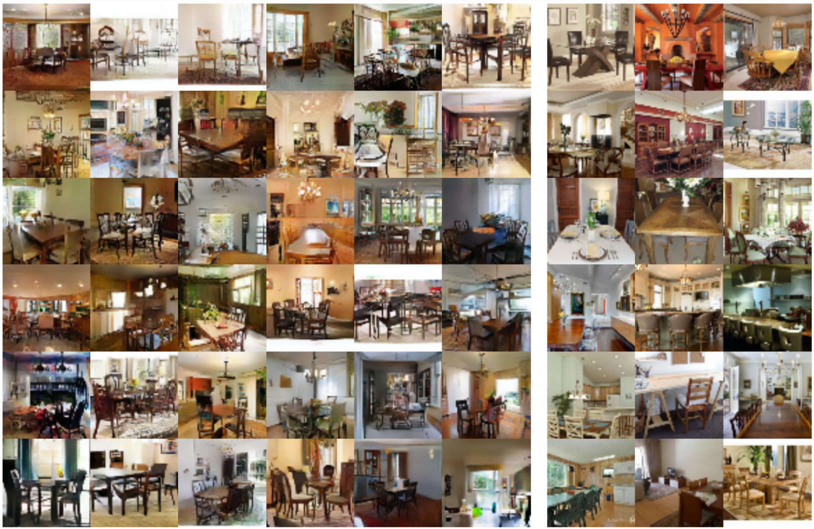
- AV-ADE: better samples, worse likelihood
- Temperature annealing allows Glow to trade-off the two

LSUN 64×64: Chruches (C) and Bedrooms (B). Figure from [Lucas et al., 2019]

Hybrid VAE-Flow model - Samples and Images



Hybrid VAE-Flow model - Samples and Images



LSUN 64 × 64: Dining rooms. Samples left, training images right.
Figure from [Lucas et al., 2019]

Part II

Recent advances in flow-based generative modeling

Reduced temperature sampling [Kingma and Dhariwal, 2018]

- Sample closer to the mode of the distribution

$$p_\tau(\mathbf{x}) \propto p(\mathbf{x})^{1/\tau} \tag{10}$$

Reduced temperature sampling [Kingma and Dhariwal, 2018]

- Sample closer to the mode of the distribution

$$p_\tau(\mathbf{x}) \propto p(\mathbf{x})^{1/\tau} \tag{10}$$

  - Approaches mode of $p(\mathbf{x})$ as $\tau \to 0$
  - Approaches uniform as $\tau \to \infty$

Reduced temperature sampling [Kingma and Dhariwal, 2018]

- Sample closer to the mode of the distribution

$$p_\tau(\mathbf{x}) \propto p(\mathbf{x})^{1/\tau} \tag{10}$$

  - Approaches mode of $p(\mathbf{x})$ as $\tau \to 0$
  - Approaches uniform as $\tau \to \infty$

- Modifies the flow in non-trivial manner

$$\ln p_\tau(\mathbf{x}) \pm \tau^{-1} \ln p_Y(f(\mathbf{x})) + \tau^{-1} \ln |\det(J_f(x))| \tag{11}$$

Reduced temperature sampling [Kingma and Dhariwal, 2018]

- Sample closer to the mode of the distribution

$$p_\tau(\mathbf{x}) \propto p(\mathbf{x})^{1/\tau} \tag{10}$$

  - Approaches mode of $p(\mathbf{x})$ as $\tau \to 0$
  - Approaches uniform as $\tau \to \infty$
- Modifies the flow in non-trivial manner

$$\ln p_\tau(\mathbf{x}) \pm \tau^{-1} \ln p_Y(f(\mathbf{x})) + \tau^{-1} \ln |\det(J_f(x))| \tag{11}$$

- Unchanged flow for $p_Y(\mathbf{y}) = \mathcal{N}(y; 0, I)$ and $\det(J_f(x)) = \text{const.}$

$$p_\tau(\mathbf{x}) \propto \mathcal{N}(f(\mathbf{x}); 0, \tau I) \tag{12}$$

Reduced temperature sampling [Kingma and Dhariwal, 2018]

- Sample closer to the mode of the distribution

$$p_\tau(\mathbf{x}) \propto p(\mathbf{x})^{1/\tau} \tag{10}$$

  - Approaches mode of $p(\mathbf{x})$ as $\tau \to 0$
  - Approaches uniform as $\tau \to \infty$

- Modifies the flow in non-trivial manner

$$\ln p_\tau(\mathbf{x}) \pm \tau^{-1} \ln p_Y(f(\mathbf{x})) + \tau^{-1} \ln |\det (J_f(x))| \tag{11}$$

- Unchanged flow for $p_Y(\mathbf{y}) = \mathcal{N}(y; 0, I)$ and $\det (J_f(x)) = \text{const.}$

$$p_\tau(\mathbf{x}) \propto \mathcal{N}(f(\mathbf{x}); 0, \tau I) \tag{12}$$

- Can sample from reduced Gaussian in latent space, and then project

Additive coupling layers

$$\mathbf{y}_1 = \mathbf{x}_1, \qquad \mathbf{y}_2 = \mathbf{x}_2 + t\left(\mathbf{x}_1\right)$$

Additive coupling layers

$$\mathbf{y}_1 = \mathbf{x}_1, \qquad \mathbf{y}_2 = \mathbf{x}_2 + t\left(\mathbf{x}_1\right)$$

- Residual layer with variable partitioning

Additive coupling layers

$$\mathbf{y}_1 = \mathbf{x}_1, \qquad \mathbf{y}_2 = \mathbf{x}_2 + t\left(\mathbf{x}_1\right)$$

- Residual layer with variable partitioning
- Can be combined with affine flow layers $\mathbf{y} = W\mathbf{x}$

Additive coupling layers

$$\mathbf{y}_1 = \mathbf{x}_1, \qquad \mathbf{y}_2 = \mathbf{x}_2 + t\left(\mathbf{x}_1\right)$$

- Residual layer with variable partitioning
- Can be combined with affine flow layers $\mathbf{y} = W\mathbf{x}$
    - Determinant constant in $\mathbf{x}$

Additive coupling layers

$$\mathbf{y}_1 = \mathbf{x}_1, \qquad \mathbf{y}_2 = \mathbf{x}_2 + t(\mathbf{x}_1)$$

- Residual layer with variable partitioning
- Can be combined with affine flow layers $\mathbf{y} = W\mathbf{x}$
    - Determinant constant in $\mathbf{x}$
    - Change of basis w.r.t. original variables

Additive coupling layers

$$\mathbf{y}_1 = \mathbf{x}_1, \qquad \mathbf{y}_2 = \mathbf{x}_2 + t(\mathbf{x}_1)$$

- Residual layer with variable partitioning
- Can be combined with affine flow layers $\mathbf{y} = W\mathbf{x}$
    - Determinant constant in $\mathbf{x}$
    - Change of basis w.r.t. original variables

Increasing temperature from left to right. Figure from [Kingma and Dhariwal, 2018].

Recipes for "efficient" invertible flows

$$\mathbf{y} = f(\mathbf{x}), \qquad J_f(x) = \frac{\partial \mathbf{y}}{\partial \mathbf{x}^\top}, \tag{13}$$

$$p_X(\mathbf{x}) = p_Y(\mathbf{y}) \times \left|\det\left(J_f(x)\right)\right| \tag{14}$$

Recipes for “efficient” invertible flows

$$\mathbf{y} = f(\mathbf{x}), \qquad J_f(x) = \frac{\partial \mathbf{y}}{\partial \mathbf{x}^\top}, \tag{13}$$

$$p_X(\mathbf{x}) = p_Y(\mathbf{y}) \times \left|\det\left(J_f(x)\right)\right| \tag{14}$$

- Training: compute $f(\mathbf{x})$ and log-determinant

Recipes for "efficient" invertible flows

$$\mathbf{y} = f(\mathbf{x}), \qquad J_f(x) = \frac{\partial \mathbf{y}}{\partial \mathbf{x}^\top}, \tag{13}$$

$$p_X(\mathbf{x}) = p_Y(\mathbf{y}) \times |\det(J_f(x))| \tag{14}$$

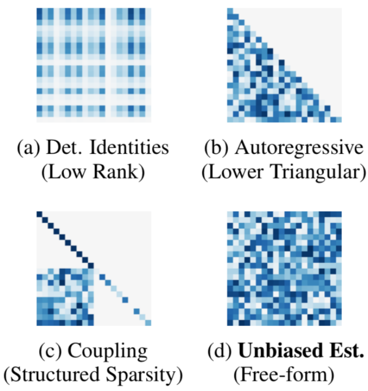
- Training: compute $f(\mathbf{x})$ and log-determinant
- Sampling: compute $f^{-1}(\mathbf{y})$

Recipes for “efficient” invertible flows

$$\mathbf{y} = f(\mathbf{x}), \qquad J_f(x) = \frac{\partial \mathbf{y}}{\partial \mathbf{x}^\top}, \tag{13}$$

$$p_X(\mathbf{x}) = p_Y(\mathbf{y}) \times \left|\det\left(J_f(x)\right)\right| \tag{14}$$

- Training: compute $f(\mathbf{x})$ and log-determinant
- Sampling: compute $f^{-1}(\mathbf{y})$



(a) Det. Identities (Low Rank)

(b) Autoregressive (Lower Triangular)

(c) Coupling (Structured Sparsity)

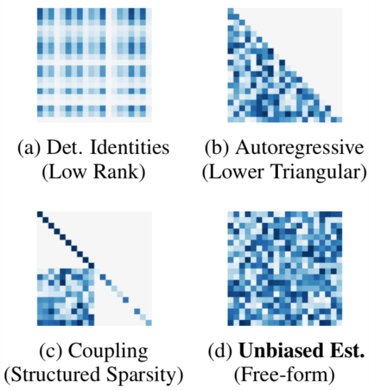
(d) **Unbiased Est.** (Free-form)

Recipes for "efficient" invertible flows

$$\mathbf{y} = f(\mathbf{x}), \qquad J_f(x) = \frac{\partial \mathbf{y}}{\partial \mathbf{x}^\top}, \tag{13}$$

$$p_X(\mathbf{x}) = p_Y(\mathbf{y}) \times |\det(J_f(x))| \tag{14}$$

- Training: compute $f(\mathbf{x})$ and log-determinant
- Sampling: compute $f^{-1}(\mathbf{y})$



(a) Planar flow [Rezende and Mohamed, 2015]

(b) Inverse Autoregressive Flow [Kingma et al., 2016]

(c) Real-NVP [Dinh et al., 2017]

(d) **Invertible ResNet** [Behrmann et al., 2019, R.Chen et al., 2019]

Invertible ResNets [Behrmann et al., 2019]

Invertible ResNets [Behrmann et al., 2019]

- Residual Networks [He et al., 2016a, He et al., 2016b]

$$y := f(x) = x + g_\theta(x) \tag{15}$$

Invertible ResNets [Behrmann et al., 2019]

- Residual Networks [He et al., 2016a, He et al., 2016b]

$$y := f(x) = x + g_\theta(x) \tag{15}$$

- Improves gradient propagation in very deep networks

Invertible ResNets [Behrmann et al., 2019]

- Residual Networks [He et al., 2016a, He et al., 2016b]

$$y := f(x) = x + g_\theta(x) \tag{15}$$

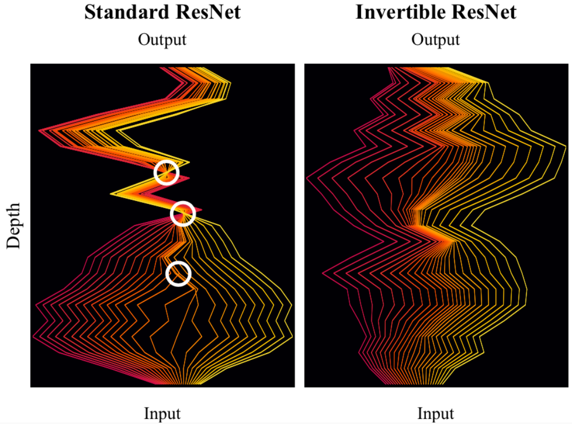
- Improves gradient propagation in very deep networks
- State of the art across many tasks, including vision CNNs

Invertible ResNets [Behrmann et al., 2019]

- Residual Networks [He et al., 2016a, He et al., 2016b]

$$y := f(x) = x + g_\theta(x) \tag{15}$$

- Improves gradient propagation in very deep networks
- State of the art across many tasks, including vision CNNs



Invertible ResNets

- Residual Networks [He et al., 2016a, He et al., 2016b]

$$y := f(x) = x + g_\theta(x) \tag{16}$$

Invertible ResNets

- Residual Networks [He et al., 2016a, He et al., 2016b]

$$y := f(x) = x + g_\theta(x) \tag{16}$$

- ResNets are invertible if $Lip(g_\theta) < 1$, **i.e**.

$$||g_\theta(x_1) - g_\theta(x_2)||_2^2 \leq ||x_1 - x_2||_2^2 \tag{17}$$

Invertible ResNets

- Residual Networks [He et al., 2016a, He et al., 2016b]

$$y := f(x) = x + g_\theta(x) \tag{16}$$

- ResNets are invertible if $Lip(g_\theta) < 1$, **i.e.**

$$||g_\theta(x_1) - g_\theta(x_2)||_2^2 \leq ||x_1 - x_2||_2^2 \tag{17}$$

- Inverse can be computed as fixed-point

$$x^0 := y, \tag{18}$$

$$x^{i+1} := y - g_\theta(x^i) \tag{19}$$

Invertible ResNets

- Residual Networks [He et al., 2016a, He et al., 2016b]

$$y := f(x) = x + g_\theta(x) \tag{16}$$

- ResNets are invertible if $Lip(g_\theta) < 1$, **i.e.**

$$||g_\theta(x_1) - g_\theta(x_2)||_2^2 \leq ||x_1 - x_2||_2^2 \tag{17}$$

- Inverse can be computed as fixed-point

$$x^0 := y, \tag{18}$$

$$x^{i+1} := y - g_\theta(x^i) \tag{19}$$

- Unbiased determinant estimator [R.Chen et al., 2019]

Invertible ResNets

- Residual Networks [He et al., 2016a, He et al., 2016b]

$$y := f(x) = x + g_\theta(x) \tag{16}$$

- ResNets are invertible if $Lip(g_\theta) < 1$, **i.e**.

$$||g_\theta(x_1) - g_\theta(x_2)||_2^2 \leq ||x_1 - x_2||_2^2 \tag{17}$$

- Inverse can be computed as fixed-point

$$x^0 := y, \tag{18}$$

$$x^{i+1} := y - g_\theta(x^i) \tag{19}$$

- Unbiased determinant estimator [R.Chen et al., 2019]
- Possible to use ResNet for flow-based generative model

Generative modeling with invertible ResNets

$$f(\mathbf{x}) = \mathbf{x} + g_\theta(\mathbf{x}) \tag{20}$$

Generative modeling with invertible ResNets

$$f(\mathbf{x}) = \mathbf{x} + g_{\theta}(\mathbf{x}) \tag{20}$$

- All variables updates in every flow step, unlike variable partitioning-scheme in Real-NVP

Generative modeling with invertible ResNets

$$f(\mathbf{x}) = \mathbf{x} + g_\theta(\mathbf{x}) \tag{20}$$

- All variables updates in every flow step,
  unlike variable partitioning-scheme in Real-NVP
- Faster "mixing" between variables

Generative modeling with invertible ResNets

$$f(\mathbf{x}) = \mathbf{x} + g_\theta(\mathbf{x}) \tag{20}$$

- All variables updates in every flow step, unlike variable partitioning-scheme in Real-NVP
- Faster "mixing" between variables

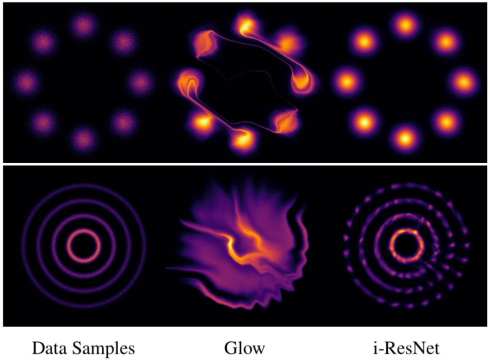


Figure from [Behrmann et al., 2019]

Invertible ResNets [Behrmann et al., 2019]

- Hybrid discriminative-generative training

$$L = \lambda \ln p(x) + \ln p(y|x) \tag{21}$$

Invertible ResNets [Behrmann et al., 2019]

- Hybrid discriminative-generative training

$$L = \lambda \ln p(x) + \ln p(y|x) \tag{21}$$

- Network fully invertible,
  until last linear classifier that projects on the label space

Invertible ResNets [Behrmann et al., 2019]

- Hybrid discriminative-generative training

$$L = \lambda \ln p(x) + \ln p(y|x) \tag{21}$$

- Network fully invertible,
  until last linear classifier that projects on the label space

	$\lambda = 0$	$\lambda = 1/D$		$\lambda = 1$	
Block Type	Acc↑	BPD↓	Acc↑	BPD↓	Acc↑
Coupling	89.77%	4.30	87.58%	3.54	67.62%
+ 1×1 Conv	90.82%	4.09	87.96%	3.47	67.38%
Residual	**91.78%**	**3.62**	**90.47%**	**3.39**	**70.32%**

Results on CIFAR-10 from [R.Chen et al., 2019]

Part III

Stabilizing GAN training

A discussion on the GAN training loss

A discussion on the GAN training loss

- Recall divergence measures between distributions

A discussion on the GAN training loss

- Recall divergence measures between distributions
- Kullback-Leibler divergence: maximum likelihood training
  - Infinite if q (model) has a zero in the support of p (data)

$$D_{KL}(p||q) = \int_x p(x)\big[\ln q(x) - \ln p(x)\big] \tag{22}$$

A discussion on the GAN training loss

- Recall divergence measures between distributions
- Kullback-Leibler divergence: maximum likelihood training
  - Infinite if q (model) has a zero in the support of p (data)

$$D_{KL}(p||q) = \int_x p(x)\big[\ln q(x) - \ln p(x)\big] \tag{22}$$

- Jensen-Shannon divergence: idealized loss approximated by the discriminator
  - Symmetric KL to mixture of p and q

$$D_{JS}(p||q) = \frac{1}{2} D_{KL}\left(p\middle|\middle|\frac{p+q}{2}\right) + \frac{1}{2} D_{KL}\left(q\middle|\middle|\frac{p+q}{2}\right) \tag{23}$$

A discussion on the GAN training loss

- Training loss for the Discriminator:

$$V(\phi, \theta) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\ln D_\phi(x)] + \mathbb{E}_{z \sim p(z)}[\ln(1 - D_\phi(f_\theta(z)))] \quad (24)$$

A discussion on the GAN training loss

- Training loss for the Discriminator:

$$V(\phi, \theta) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\ln D_\phi(x)] + \mathbb{E}_{z \sim p(z)}[\ln(1 - D_\phi(f_\theta(z)))]$$

A discussion on the GAN training loss

- Training loss for the Discriminator:

$$V(\phi, \theta) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\ln D_\phi(x)] + \mathbb{E}_{z \sim p(z)}[\ln(1 - D_\phi(f_\theta(z)))]$$

- Approximates the ideal loss:

$$D_{JS}(p||q) = \frac{1}{2} D_{KL}\left(p \middle\| \frac{p+q}{2}\right) + \frac{1}{2} D_{KL}\left(q \middle\| \frac{p+q}{2}\right) \qquad (24)$$

A discussion on the GAN training loss

- Training loss for the Discriminator:

$$V(\phi, \theta) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\ln D_\phi(x)] + \mathbb{E}_{z \sim p(z)}[\ln(1 - D_\phi(f_\theta(z)))]$$

- Approximates the ideal loss:

$$D_{JS}(p||q) = \frac{1}{2} D_{KL}\left(p \middle\| \frac{p+q}{2}\right) + \frac{1}{2} D_{KL}\left(q \middle\| \frac{p+q}{2}\right) \quad (24)$$

A discussion on the GAN training loss

- Training loss for the Discriminator:

$$V(\phi, \theta) = \mathbb{E}_{x \sim p_{\mathrm{data}}(x)}[\ln D_\phi(x)] + \mathbb{E}_{z \sim p(z)}[\ln(1 - D_\phi(f_\theta(z)))]$$

- Approximates the ideal loss:

$$D_{JS}(p||q) = \frac{1}{2} D_{KL}\left(p \middle\| \frac{p+q}{2}\right) + \frac{1}{2} D_{KL}\left(q \middle\| \frac{p+q}{2}\right) \tag{24}$$

- The blue term is independent from the model p_θ, and disapears when differentiating

A discussion on the GAN training loss

- Training loss for the Discriminator:

$$V(\phi, \theta) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\ln D_\phi(x)] + \mathbb{E}_{z \sim p(z)}[\ln(1 - D_\phi(f_\theta(z)))]$$

- Approximates the ideal loss:

$$D_{JS}(p||q) = \frac{1}{2} D_{KL}\left(p \middle\| \frac{p+q}{2}\right) + \frac{1}{2} D_{KL}\left(q \middle\| \frac{p+q}{2}\right) \quad (24)$$

- The blue term is independent from the model p_θ, and disapears when differentiating
- The generator is trained on the red term

Quality driven training

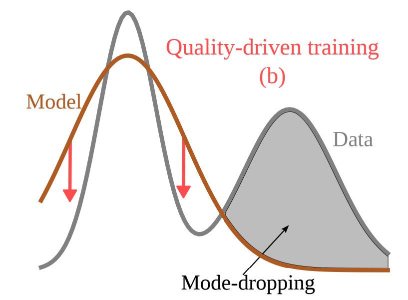
- Training loss for the generator: $D_{KL}\left(q \middle\| \frac{p+q}{2}\right)$

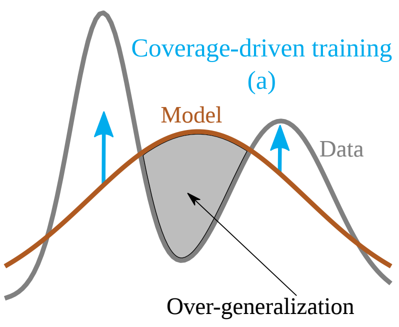
Quality driven training

- Training loss for the generator: $D_{KL}\left(q \middle|\middle| \frac{p+q}{2}\right)$
- It is an integral on q, opposite to maximum-likelihood estimation

Quality driven training

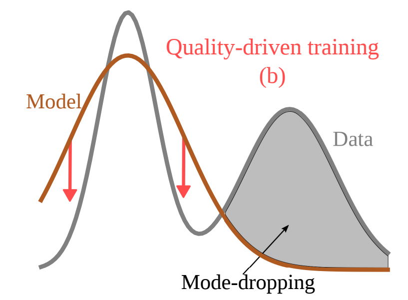
- Training loss for the generator: $D_{KL}\left(q \middle\| \frac{p+q}{2}\right)$
- It is an integral on q, opposite to maximum-likelihood estimation



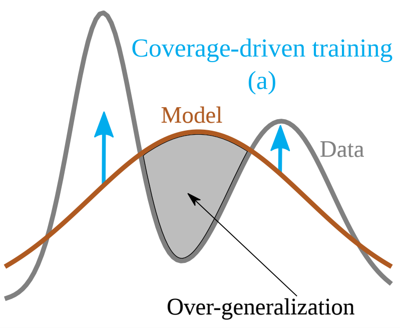


Quality driven training

- Training loss for the generator: $D_{KL}\left(q \middle\| \frac{p+q}{2}\right)$
- It is an integral on q, opposite to maximum-likelihood estimation



$\frac{1}{2} D_{KL}\left(q \middle\| \frac{p+q}{2}\right)$



$\frac{1}{2} D_{KL}\left(p \middle\| \frac{p+q}{2}\right)$

Why is GAN training is difficult in practice? [Arjovsky et al., 2017]

Why is GAN training is difficult in practice? [Arjovsky et al., 2017]

1. Strong discriminator leads to vanishing gradients of $\mathbb{E}_{p_z}[\ln(1 - D(G(z)))]$ w.r.t. generator
    - Happens early in training with poor generator

Why is GAN training is difficult in practice? [Arjovsky et al., 2017]

1. Strong discriminator leads to vanishing gradients of $\mathbb{E}_{p_z}[\ln(1 - D(G(z)))]$ w.r.t. generator
    - Happens early in training with poor generator
    - Tuning of capacity and training regime of discriminator

Why is GAN training is difficult in practice? [Arjovsky et al., 2017]

1. Strong discriminator leads to vanishing gradients of $\mathbb{E}_{p_z}[\ln(1 - D(G(z)))]$ w.r.t. generator
    - Happens early in training with poor generator
    - Tuning of capacity and training regime of discriminator

2. Minimizing $-\mathbb{E}_{p_z}[\ln(D(G(z)))]$ instead to boost gradient

Why is GAN training is difficult in practice? [Arjovsky et al., 2017]

1. Strong discriminator leads to vanishing gradients of $\mathbb{E}_{p_z}[\ln(1 - D(G(z)))]$ w.r.t. generator
   - Happens early in training with poor generator
   - Tuning of capacity and training regime of discriminator
2. Minimizing $-\mathbb{E}_{p_z}[\ln(D(G(z)))]$ instead to boost gradient
   - Optimizes $KL(p_G || p_{\text{data}}) - 2JS(p_G || p_{\text{data}})$

Why is GAN training is difficult in practice? [Arjovsky et al., 2017]

1. Strong discriminator leads to vanishing gradients of $\mathbb{E}_{p_z}[\ln(1 - D(G(z)))]$ w.r.t. generator
    - Happens early in training with poor generator
    - Tuning of capacity and training regime of discriminator

2. Minimizing $-\mathbb{E}_{p_z}[\ln(D(G(z)))]$ instead to boost gradient
    - Optimizes $KL(p_G||p_{\mathrm{data}}) - 2JS(p_G||p_{\mathrm{data}})$
    - Wrong sign in the JS divergence

Why is GAN training is difficult in practice? [Arjovsky et al., 2017]

1. Strong discriminator leads to vanishing gradients of $\mathbb{E}_{p_z}[\ln(1 - D(G(z)))]$ w.r.t. generator
    - Happens early in training with poor generator
    - Tuning of capacity and training regime of discriminator

2. Minimizing $-\mathbb{E}_{p_z}[\ln(D(G(z)))]$ instead to boost gradient
    - Optimizes $KL(p_G||p_{\mathrm{data}}) - 2JS(p_G||p_{\mathrm{data}})$
    - Wrong sign in the JS divergence
    - Same stable points in the minimax optimization

Why is GAN training is difficult in practice? [Arjovsky et al., 2017]

1. Strong discriminator leads to vanishing gradients of $\mathbb{E}_{p_z}[\ln(1 - D(G(z)))]$ w.r.t. generator
   - Happens early in training with poor generator
   - Tuning of capacity and training regime of discriminator
2. Minimizing $-\mathbb{E}_{p_z}[\ln(D(G(z)))]$ instead to boost gradient
   - Optimizes $KL(p_G||p_{\mathrm{data}}) - 2JS(p_G||p_{\mathrm{data}})$
   - Wrong sign in the JS divergence
   - Same stable points in the minimax optimization
   - Helps, but problem remains: as D_ϕ becomes strong, gradients vanish

Question:

Can we think of a better 'ideal loss'?

Wasserstein or "earth-mover" distance

- Consider joint distribution $\gamma(x, y)$
  with marginals $p(x) = \gamma(x)$ and $q(y) = \gamma(y)$

Wasserstein or "earth-mover" distance

- Consider joint distribution $\gamma(x, y)$
  with marginals $p(x) = \gamma(x)$ and $q(y) = \gamma(y)$
- Conditional $\gamma(y|x)$ "moves mass" to transform $p(\cdot)$ into $q(\cdot)$

Wasserstein or "earth-mover" distance

- Consider joint distribution $\gamma(x, y)$
  with marginals $p(x) = \gamma(x)$ and $q(y) = \gamma(y)$
- Conditional $\gamma(y|x)$ "moves mass" to transform $p(\cdot)$ into $q(\cdot)$
- Cost associated with a given transformation

$$T(\gamma) = \int_{x,y} \gamma(x, y)\, ||x - y|| = \int_x p(x) \int_y \gamma(y|x)\, ||x - y||$$

Wasserstein or "earth-mover" distance

- Consider joint distribution $\gamma(x, y)$
  with marginals $p(x) = \gamma(x)$ and $q(y) = \gamma(y)$
- Conditional $\gamma(y|x)$ "moves mass" to transform $p(\cdot)$ into $q(\cdot)$
- Cost associated with a given transformation

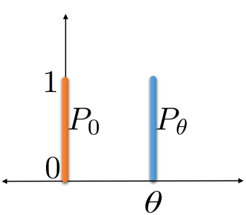
$$T(\gamma) = \int_{x,y} \gamma(x, y) \, ||x - y|| = \int_x p(x) \int_y \gamma(y|x) \, ||x - y||$$

- Wasserstein distance is the cost of optimal transformation

$$D_{WS}(p||q) = \inf_{\gamma \in \Gamma(p,q)} T(\gamma) \tag{25}$$

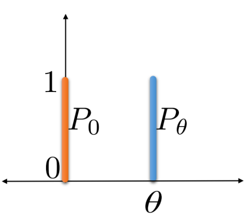
Distributions with low dimensional support

- Simple example: support on lines in $\mathbb{R}^2$
  - p_0 uniform on $x_2 \in [0, 1]$ for $x_1 = 0$
  - p_θ uniform on $x_2 \in [0, 1]$ for $x_1 = \theta$



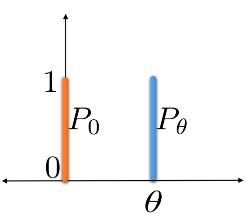
Distributions with low dimensional support

- Simple example: support on lines in $\mathbb{R}^2$
  - p_0 uniform on $x_2 \in [0, 1]$ for $x_1 = 0$
  - p_θ uniform on $x_2 \in [0, 1]$ for $x_1 = \theta$
- All measures zero for $\theta = 0$, but for $\theta \neq 0$



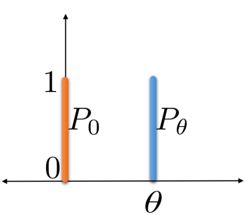
Distributions with low dimensional support

- Simple example: support on lines in $\mathbb{R}^2$
  - p_0 uniform on $x_2 \in [0, 1]$ for $x_1 = 0$
  - p_θ uniform on $x_2 \in [0, 1]$ for $x_1 = \theta$
- All measures zero for $\theta = 0$, but for $\theta \neq 0$
  - $D_{KL}(p_0 || p_\theta) = \infty$



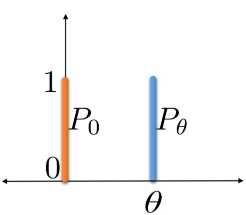
Distributions with low dimensional support

- Simple example: support on lines in $\mathbb{R}^2$
  - p_0 uniform on $x_2 \in [0, 1]$ for $x_1 = 0$
  - p_θ uniform on $x_2 \in [0, 1]$ for $x_1 = \theta$
- All measures zero for $\theta = 0$, but for $\theta \neq 0$
  - $D_{KL}(p_0 || p_\theta) = \infty$
  - $D_{JS}(p_0 || p_\theta) = \ln 2$



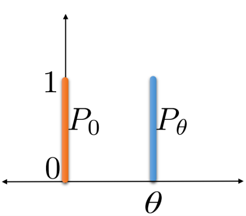
Distributions with low dimensional support

- Simple example: support on lines in $\mathbb{R}^2$
  - p_0 uniform on $x_2 \in [0, 1]$ for $x_1 = 0$
  - p_θ uniform on $x_2 \in [0, 1]$ for $x_1 = \theta$
- All measures zero for $\theta = 0$, but for $\theta \neq 0$
  - $D_{KL}(p_0 || p_\theta) = \infty$
  - $D_{JS}(p_0 || p_\theta) = \ln 2$
  - $D_{WS}(p_0 || p_\theta) = |\theta|$



Distributions with low dimensional support

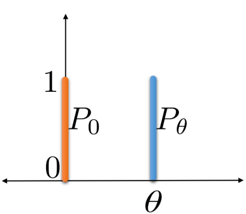
- Simple example: support on lines in $\mathbb{R}^2$
  - p_0 uniform on $x_2 \in [0,1]$ for $x_1 = 0$
  - p_θ uniform on $x_2 \in [0,1]$ for $x_1 = \theta$
- All measures zero for $\theta = 0$, but for $\theta \neq 0$
  - $D_{KL}(p_0 || p_\theta) = \infty$
  - $D_{JS}(p_0 || p_\theta) = \ln 2$
  - $D_{WS}(p_0 || p_\theta) = |\theta|$



- Wasserstein based on proximity of support

Distributions with low dimensional support

- Simple example: support on lines in $\mathbb{R}^2$
  - p_0 uniform on $x_2 \in [0,1]$ for $x_1 = 0$
  - p_θ uniform on $x_2 \in [0,1]$ for $x_1 = \theta$

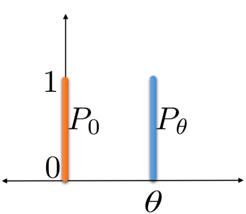


- All measures zero for $\theta = 0$, but for $\theta \neq 0$
  - $D_{KL}(p_0 || p_\theta) = \infty$
  - $D_{JS}(p_0 || p_\theta) = \ln 2$
  - $D_{WS}(p_0 || p_\theta) = |\theta|$

- Wasserstein based on proximity of support
- JS and KL based on overlap of support

Distributions with low dimensional support

- Simple example: support on lines in $\mathbb{R}^2$
  - p_0 uniform on $x_2 \in [0, 1]$ for $x_1 = 0$
  - p_θ uniform on $x_2 \in [0, 1]$ for $x_1 = \theta$

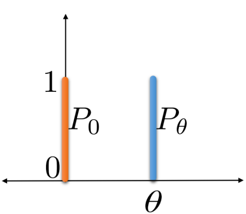


- All measures zero for $\theta = 0$, but for $\theta \neq 0$
  - $D_{KL}(p_0||p_\theta) = \infty$
  - $D_{JS}(p_0||p_\theta) = \ln 2$
  - $D_{WS}(p_0||p_\theta) = |\theta|$

- Wasserstein based on proximity of support
- JS and KL based on overlap of support
  - In general measure zero overlap with low dim. supports

Distributions with low dimensional support

- Simple example: support on lines in $\mathbb{R}^2$
  - p_0 uniform on $x_2 \in [0, 1]$ for $x_1 = 0$
  - p_θ uniform on $x_2 \in [0, 1]$ for $x_1 = \theta$
- All measures zero for $\theta = 0$, but for $\theta \neq 0$
  - $D_{KL}(p_0||p_\theta) = \infty$
  - $D_{JS}(p_0||p_\theta) = \ln 2$
  - $D_{WS}(p_0||p_\theta) = |\theta|$



- Wasserstein based on proximity of support
- JS and KL based on overlap of support
  - In general measure zero overlap with low dim. supports
  - GAN has support with dimension of latent variable **z**

Wasserstein GAN

- Dual formulation of Wasserstein distance

$$D_{WS}(p_data||q) = \inf_{\gamma \in \Gamma(p,q)} T(\gamma) \quad (26)$$

$$= \frac{1}{k} \max_{||D||_L \leq k} \mathbb{E}_{p_{\mathrm{data}}}[D(\mathbf{x})] - \mathbb{E}_{p_z}[D(G(\mathbf{z}))] \quad (27)$$

Wasserstein GAN

- Dual formulation of Wasserstein distance

$$D_{WS}(p_d ata||q) = \inf_{\gamma \in \Gamma(p,q)} T(\gamma) \quad (26)$$

$$= \frac{1}{k} \max_{||D||_L \leq k} \mathbb{E}_{p_{\text{data}}}[D(\mathbf{x})] - \mathbb{E}_{p_z}[D(G(\mathbf{z}))] \quad (27)$$

1. $||.||_L$ is the lipschitz norm

Wasserstein GAN

- Dual formulation of Wasserstein distance

$$D_{WS}(p_data||q) = \inf_{\gamma \in \Gamma(p,q)} T(\gamma) \quad (26)$$

$$= \frac{1}{k} \max_{||D||_L \leq k} \mathbb{E}_{p_{\text{data}}}[D(\mathbf{x})] - \mathbb{E}_{p_z}[D(G(\mathbf{z}))] \quad (27)$$

1. $||.||_L$ is the lipschitz norm
2. In practice: restrict D to some deep net architecture

Wasserstein GAN

- Dual formulation of Wasserstein distance

$$D_{WS}(p_data||q) = \inf_{\gamma \in \Gamma(p,q)} T(\gamma) \quad (26)$$

$$= \frac{1}{k} \max_{||D||_L \leq k} \mathbb{E}_{p_{\text{data}}}[D(\mathbf{x})] - \mathbb{E}_{p_z}[D(G(\mathbf{z}))] \quad (27)$$

1. $||.||_L$ is the lipschitz norm
2. In practice: restrict D to some deep net architecture
3. Enforce Lipschitz constraint by clipping discriminator weights or penalty on gradient magnitude [Gulrajani et al., 2017a]

Wasserstein GAN

- Dual formulation of Wasserstein distance

$$D_{WS}(p_data||q) = \inf_{\gamma \in \Gamma(p,q)} T(\gamma) \quad (26)$$

$$= \frac{1}{k} \max_{||D||_L \leq k} \mathbb{E}_{p_{\mathrm{data}}}[D(\mathbf{x})] - \mathbb{E}_{p_z}[D(G(\mathbf{z}))] \quad (27)$$

1. $||.||_L$ is the lipschitz norm
2. In practice: restrict D to some deep net architecture
3. Enforce Lipschitz constraint by clipping discriminator weights or penalty on gradient magnitude [Gulrajani et al., 2017a]

- Removes log-sigmoid transformation w.r.t. normal GAN

Wasserstein GAN

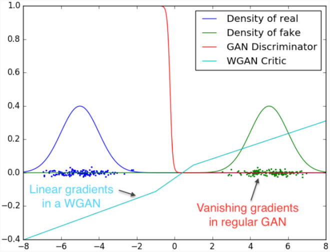
- Dual formulation of Wasserstein distance

$$D_{WS}(p_data||q) = \inf_{\gamma \in \Gamma(p,q)} T(\gamma) \quad (26)$$

$$= \frac{1}{k} \max_{||D||_L \leq k} \mathbb{E}_{p_{\mathrm{data}}}[D(\mathbf{x})] - \mathbb{E}_{p_z}[D(G(\mathbf{z}))] \quad (27)$$

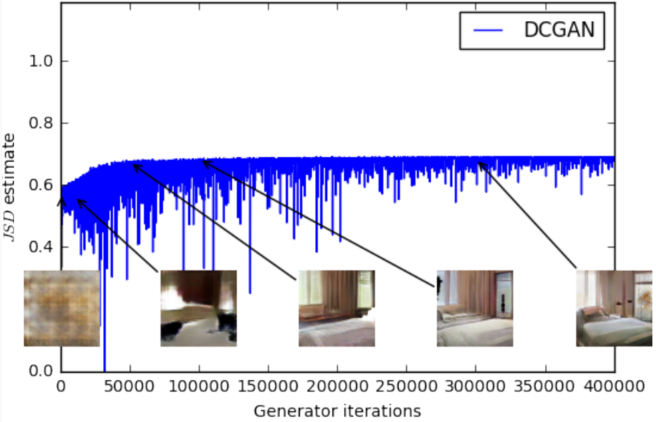
1. $||.||_L$ is the lipschitz norm
2. In practice: restrict D to some deep net architecture
3. Enforce Lipschitz constraint by clipping discriminator weights or penalty on gradient magnitude [Gulrajani et al., 2017a]

- Removes log-sigmoid transformation w.r.t. normal GAN



Experimental comparison GAN and WGAN

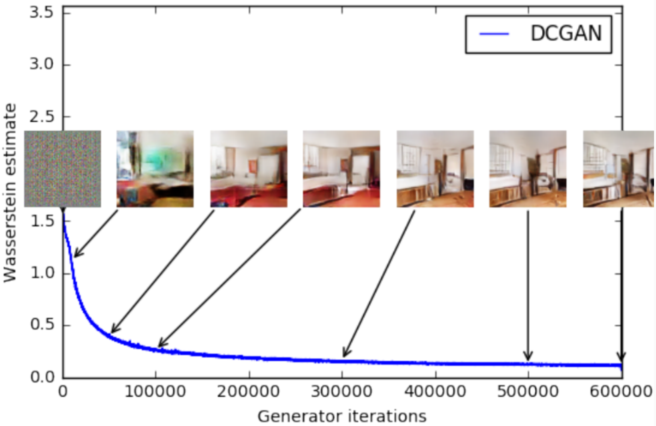
- WGAN loss may decrease in a more stable manner
- WGAN loss correlates better with sample quality



GAN

Experimental comparison GAN and WGAN

- WGAN loss may decrease in a more stable manner
- WGAN loss correlates better with sample quality



WGAN

In practice

Is this analysis relevant in practice?

In practice

Is this analysis relevant in practice?

- This analysis regards the **ideal losses** (D_{KL} VS. D_{WS})

In practice

Is this analysis relevant in practice?

- This analysis regards the **ideal losses** (D_{KL} VS. D_{WS})
- In practice, both are approximated by **similar discriminators**
  - $L_{WGAN} = \frac{1}{k} \max_{||D||_L \leq k} \mathbb{E}_{p_{\mathrm{data}}}[D(\mathbf{x})] - \mathbb{E}_{p_z}[D(G(\mathbf{z}))]$
  - $L_{GAN} = \frac{1}{k} \max_D \mathbb{E}_{p_{\mathrm{data}}}[log(D(\mathbf{x}))] - \mathbb{E}_{p_z}[log(1 - D(G(\mathbf{z})))]$

In practice

Is this analysis relevant in practice?

- This analysis regards the **ideal losses** (D_{KL} VS. D_{WS})
- In practice, both are approximated by **similar discriminators**
  - $L_{WGAN} = \frac{1}{k} \max_{||D||_L \leq k} \mathbb{E}_{p_{\text{data}}}[D(\mathbf{x})] - \mathbb{E}_{p_z}[D(G(\mathbf{z}))]$
  - $L_{GAN} = \frac{1}{k} \max_D \mathbb{E}_{p_{\text{data}}}[log(D(\mathbf{x}))] - \mathbb{E}_{p_z}[log(1 - D(G(\mathbf{z})))]$
- In practice, non-overlapping support does not break the discriminator

In practice

Is this analysis relevant in practice?

- This analysis regards the **ideal losses** (D_{KL} VS. D_{WS})
- In practice, both are approximated by **similar discriminators**
  - $L_{WGAN} = \frac{1}{k} \max_{||D||_L \leq k} \mathbb{E}_{p_{\text{data}}}[D(\mathbf{x})] - \mathbb{E}_{p_z}[D(G(\mathbf{z}))]$
  - $L_{GAN} = \frac{1}{k} \max_D \mathbb{E}_{p_{\text{data}}}[log(D(\mathbf{x}))] - \mathbb{E}_{p_z}[log(1 - D(G(\mathbf{z})))]$
- In practice, non-overlapping support does not break the discriminator
- Constraining the Lipshitz constant is a good regularizer

Is this analysis relevant in practice?

- This analysis regards the **ideal losses** (D_{KL} VS. D_{WS})
- In practice, both are approximated by **similar discriminators**
  - $L_{WGAN} = \frac{1}{k} \max_{||D||_L \leq k} \mathbb{E}_{p_{\mathrm{data}}}[D(\mathbf{x})] - \mathbb{E}_{p_z}[D(G(\mathbf{z}))]$
  - $L_{GAN} = \frac{1}{k} \max_D \mathbb{E}_{p_{\mathrm{data}}}[log(D(\mathbf{x}))] - \mathbb{E}_{p_z}[log(1 - D(G(\mathbf{z})))]$
- In practice, non-overlapping support does not break the discriminator
- Constraining the Lipshitz constant is a good regularizer
- Removing the log avoids vanishing gradients

Lipschitz continuity as a regularizer

- **Reminder:** k-Lispschitz means $|f(x) - f(y)| \leq |x - y|$
- **Reminder:** For linear functions, the largest singular value

Lipschitz continuity as a regularizer

- **Reminder:** k-Lispschitz means $|f(x) - f(y)| \leq |x - y|$
- **Reminder:** For linear functions, the largest singular value
- Lispschitz continuity now widely used, but avoid clipping

Lipschitz continuity as a regularizer

- **Reminder:** k-Lispschitz means $|f(x) - f(y)| \leq |x - y|$
- **Reminder:** For linear functions, the largest singular value
- Lispschitz continuity now widely used, but avoid clipping
- Spectral Normalization [Miyato et al., 2018]
  - Approximate the spectral norm using the power iteration method
  - Divide each weight matrix by it's spectral norm
  - Spectral norm of full network is bounded by the product of norms

Lipschitz continuity as a regularizer

- **Reminder:** k-Lispschitz means $|f(x) - f(y)| \leq |x - y|$
- **Reminder:** For linear functions, the largest singular value
- Lispschitz continuity now widely used, but avoid clipping
- Spectral Normalization [Miyato et al., 2018]
  - Approximate the spectral norm using the power iteration method
  - Divide each weight matrix by it's spectral norm
  - Spectral norm of full network is bounded by the product of norms
- Gradient penalty [Gulrajani et al., 2017a]
  - Add a penalty to the loss:

$$G_{\text{pen}} = \lambda \mathbb{E}_x[||\nabla_x D(x)||_2 - 1)^2]$$

Wrap up

- A **lot** of other losses have been develloped
- The lipschitz regularization is a widely adopted regularization
- The log is usually avoided to improve gradients when Discriminator is good.

Latent variable inference in GANs [Donahue et al., 2017]

- Vanilla GAN lacks a mechanism to infer $\mathbf{z}$ from $\mathbf{x}$

Latent variable inference in GANs [Donahue et al., 2017]

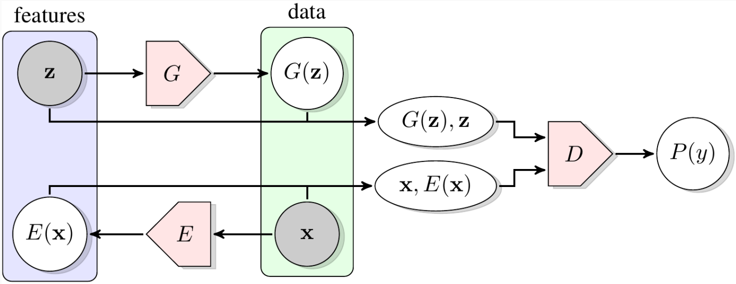
- Vanilla GAN lacks a mechanism to infer $\mathbf{z}$ from $\mathbf{x}$
- Generator: maps latent variable $\mathbf{z}$ to data point $\mathbf{x}$

Latent variable inference in GANs [Donahue et al., 2017]

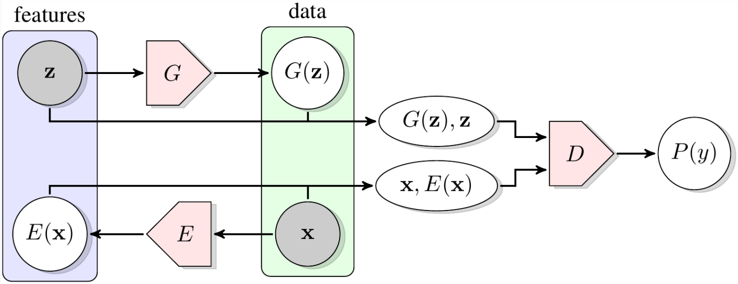
- Vanilla GAN lacks a mechanism to infer $\mathbf{z}$ from $\mathbf{x}$
- Generator: maps latent variable $\mathbf{z}$ to data point $\mathbf{x}$
- Encoder: infers latent representation $\mathbf{z}$ from data point $\mathbf{x}$

Latent variable inference in GANs [Donahue et al., 2017]

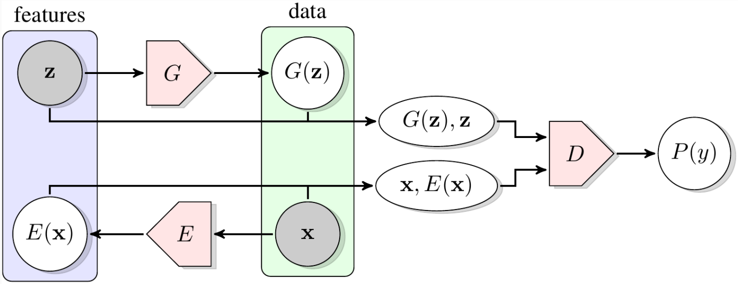
- Vanilla GAN lacks a mechanism to infer $\mathbf{z}$ from $\mathbf{x}$
- Generator: maps latent variable $\mathbf{z}$ to data point $\mathbf{x}$
- Encoder: infers latent representation $\mathbf{z}$ from data point $\mathbf{x}$



Induced joint distributions over $(\mathbf{x}, \mathbf{z})$

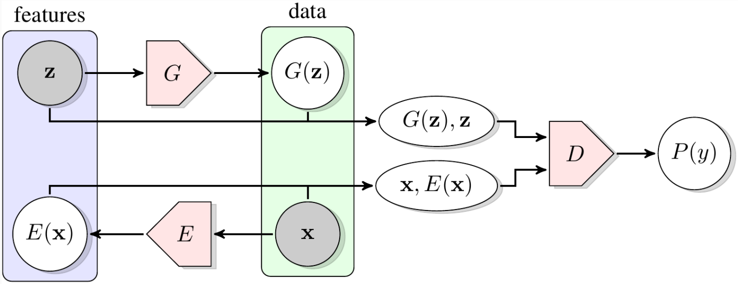


Induced joint distributions over $(\mathbf{x}, \mathbf{z})$



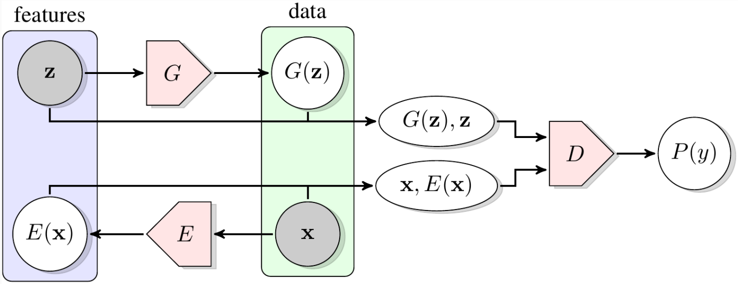
- Generator: $p_G(\mathbf{x}, \mathbf{z}) = p_{\mathbf{z}}(\mathbf{z}) \, \delta \left(\mathbf{x} - G(\mathbf{z}) \right)$

Induced joint distributions over $(\mathbf{x}, \mathbf{z})$



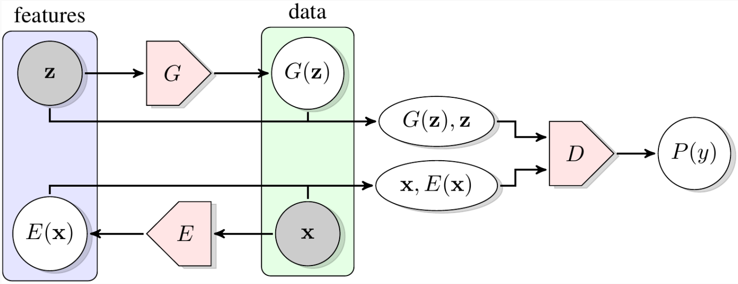
- Generator: $p_G(\mathbf{x}, \mathbf{z}) = p_{\mathbf{z}}(\mathbf{z})\, \delta\left(\mathbf{x} - G(\mathbf{z})\right)$
- Encoder: $p_E(\mathbf{x}, \mathbf{z}) = p_{\text{data}}(\mathbf{x})\, \delta\left(\mathbf{z} - E(\mathbf{x})\right)$

Induced joint distributions over $(\mathbf{x}, \mathbf{z})$



- Generator: $p_G(\mathbf{x}, \mathbf{z}) = p_{\mathbf{z}}(\mathbf{z})\ \delta\left(\mathbf{x} - G(\mathbf{z})\right)$
- Encoder: $p_E(\mathbf{x}, \mathbf{z}) = p_{\text{data}}(\mathbf{x})\ \delta\left(\mathbf{z} - E(\mathbf{x})\right)$
- Discriminator: pair $(\mathbf{x}, \mathbf{z})$ completed by generator or encoder?

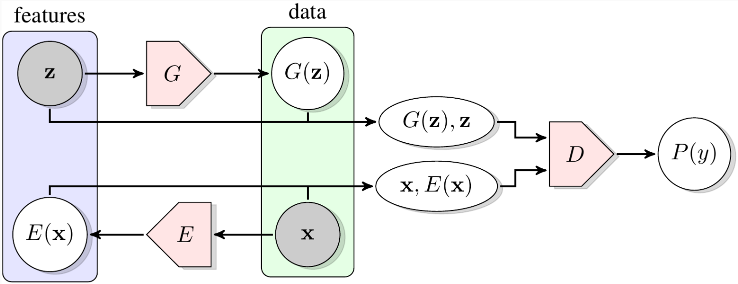
Bidirectional GANs [Donahue et al., 2017]



$$V(D, E, G) = \mathbb{E}_{p_{\text{data}}}[\ln D(\mathbf{x}, E(\mathbf{x}))] + \mathbb{E}_{p(\mathbf{z})}[\ln(1 - D(G(\mathbf{z}), \mathbf{z}))]$$

$$\min_{G,E} \max_{D} V(D, E, G)$$

Bidirectional GANs [Donahue et al., 2017]



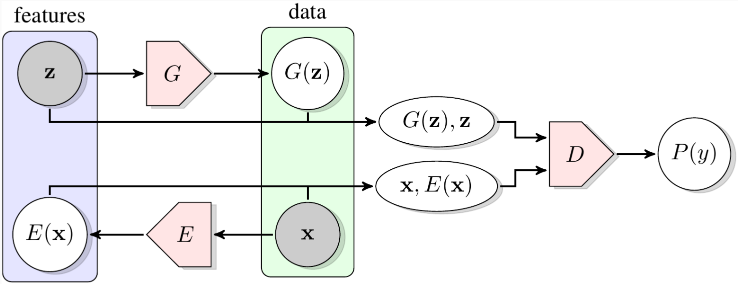
$$V(D, E, G) = \mathbb{E}_{p_{\text{data}}}[\ln D(\mathbf{x}, E(\mathbf{x}))] + \mathbb{E}_{p(\mathbf{z})}[\ln(1 - D(G(\mathbf{z}), \mathbf{z}))]$$

$$\min_{G,E} \max_{D} V(D, E, G)$$

- For optimal discriminator objective equals JS divergence

$$\max_{D} V(D, E, G) = 2D_{JS}\left(p_E(\mathbf{x}, \mathbf{z}) || p_G(\mathbf{x}, \mathbf{z})\right) - \ln 4$$

Bidirectional GANs [Donahue et al., 2017]



$$V(D, E, G) = \mathbb{E}_{p_{\text{data}}}[\ln D(\mathbf{x}, E(\mathbf{x}))] + \mathbb{E}_{p(\mathbf{z})}[\ln(1 - D(G(\mathbf{z}), \mathbf{z}))]$$

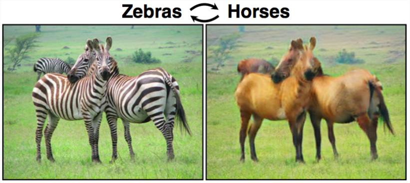
$$\min_{G,E} \max_{D} V(D, E, G)$$

- For optimal discriminator objective equals JS divergence

$$\max_{D} V(D, E, G) = 2D_{JS}\left(p_E(\mathbf{x}, \mathbf{z}) || p_G(\mathbf{x}, \mathbf{z})\right) - \ln 4$$

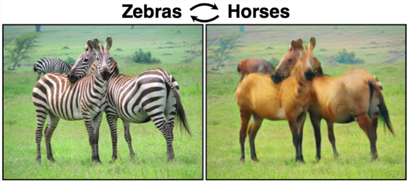
- At optimum G and E are each others inverse

Unpaired image-to-image translation [Zhu et al., 2017]



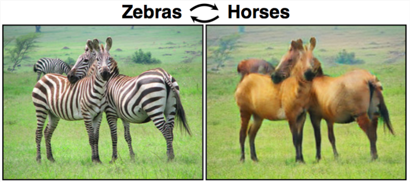
- Learn 2-way mapping between different image domains

Unpaired image-to-image translation [Zhu et al., 2017]



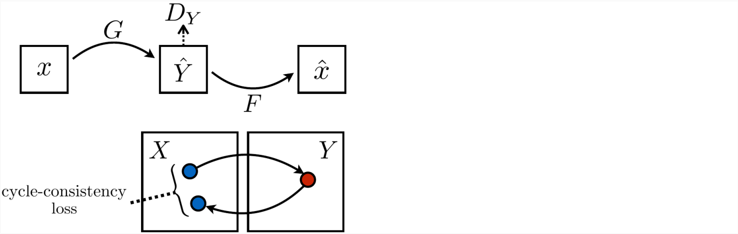
- Learn 2-way mapping between different image domains
- Without using supervised aligned training samples

Unpaired image-to-image translation [Zhu et al., 2017]

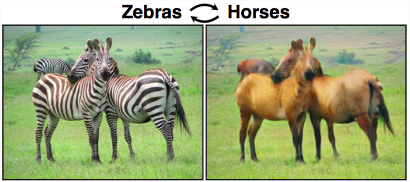


- Learn 2-way mapping between different image domains
- Without using supervised aligned training samples

1. Discriminator ensures realistic samples in each domain

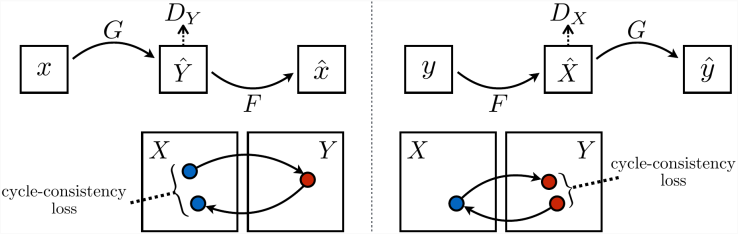


Unpaired image-to-image translation [Zhu et al., 2017]

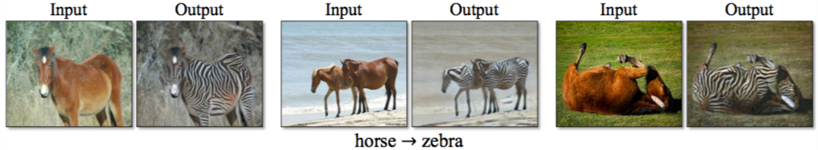


- Learn 2-way mapping between different image domains
- Without using supervised aligned training samples

1. Discriminator ensures realistic samples in each domain
2. Cycle-consistency loss ensures alignment

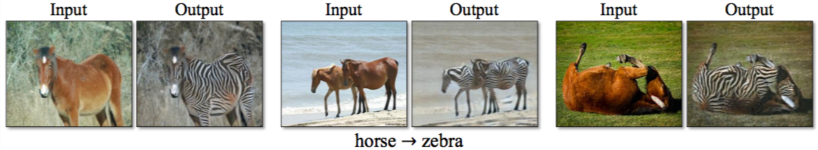


Some successful examples



Some successful examples

- Without using any supervised/aligned examples!



Some successful examples

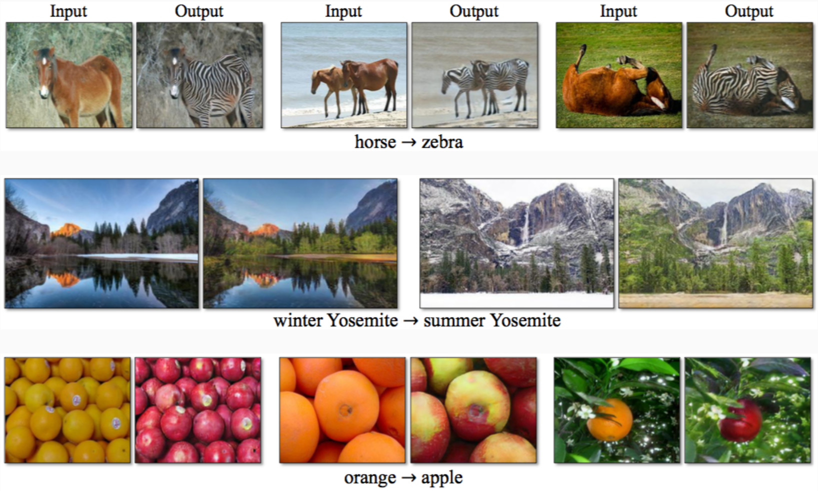
- Without using any supervised/aligned examples!

horse → zebra

winter Yosemite → summer Yosemite

Some successful examples

- Without using any supervised/aligned examples!



And a failure case

Wrap-up on GANs

Summary of what we discussed

- Improved losses using lipschitz constraints, inspired by earth-mover distance
- Adversarially trained inference networks.
- Style transfer

Thank you!

Jakob Verbeek
INRIA, Grenoble, France

jakob.verbeek@inria.fr

References i

Arjovsky, M., Chintala, S., and Bottou, L. (2017).
Wasserstein generative adversarial networks.
In *ICML*.

Behrmann, J., Grathwohl, W., Chen, R., Duvenaud, D., and Jacobsen, J.-H. (2019).
Invertible residual networks.
In *ICML*.

Burda, Y., Salakhutdinov, R., and Grosse, R. (2016).
Importance weighted autoencoders.
In *ICLR*.

Chen, X., Kingma, D., Salimans, T., Duan, Y., Dhariwal, P., Schulman, J., Sutskever, I., and Abbeel, P. (2017).
Variational lossy autoencoder.
In *ICLR*.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2017).
Density estimation using real NVP.
In *ICLR*.

Donahue, J., Krähenbühl, P., and Darrell, T. (2017).
Adversarial feature learning.
In *ICLR*.

References ii

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. (2017a).
Improved training of Wasserstein GANs.
In *NeurIPS*.

Gulrajani, I., Kumar, K., Ahmed, F., Taiga, A. A., Visin, F., Vazquez, D., and Courville, A. (2017b).
PixelVAE: A latent variable model for natural images.
In *ICLR*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016a).
Deep residual learning for image recognition.
In *CVPR*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016b).
Identity mappings in deep residual networks.
In *ECCV*.

Kingma, D. and Dhariwal, P. (2018).
Glow: Generative flow with invertible 1x1 convolutions.
In *NeurIPS*.

Kingma, D., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. (2016).
Improved variational inference with inverse autoregressive flow.
In *NeurIPS*.

References iii

Lucas, T., Shmelkov, K., Alahari, K., Schmid, C., and Verbeek, J. (2019).
Adaptive density estimation for generative models.
In *NeurIPS*.

Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. (2018).
Spectral normalization for generative adversarial networks.
In *ICLR*.

R.Chen, Behrmann, J., Duvenaud, D., and Jacobsen, J.-H. (2019).
Residual flows for invertible generative modeling.
In *NeurIPS*.

Rezende, D. and Mohamed, S. (2015).
Variational inference with normalizing flows.
In *ICML*.

Zhu, J.-Y., Park, T., Isola, P., and Efros, A. (2017).
Unpaired image-to-image translation using cycle-consistent adversarial networks.
In *ICCV*.