Generative Models:
Recent Progresses and Applications

CEDL 2016

Topics

- Inference?
  - Inverse mapping to the latent space for GAN
- Extensions and applications

Adversarially Learned Inference

Vincent Dumoulin[*]
vincent.dumoulin@umontreal.ca

Ishmael Belghazi[*]
ishmael.belghazi@gmail.com

Ben Poole [†]
poole@cs.stanford.edu

Alex Lamb[*]
alex6200@gmail.com

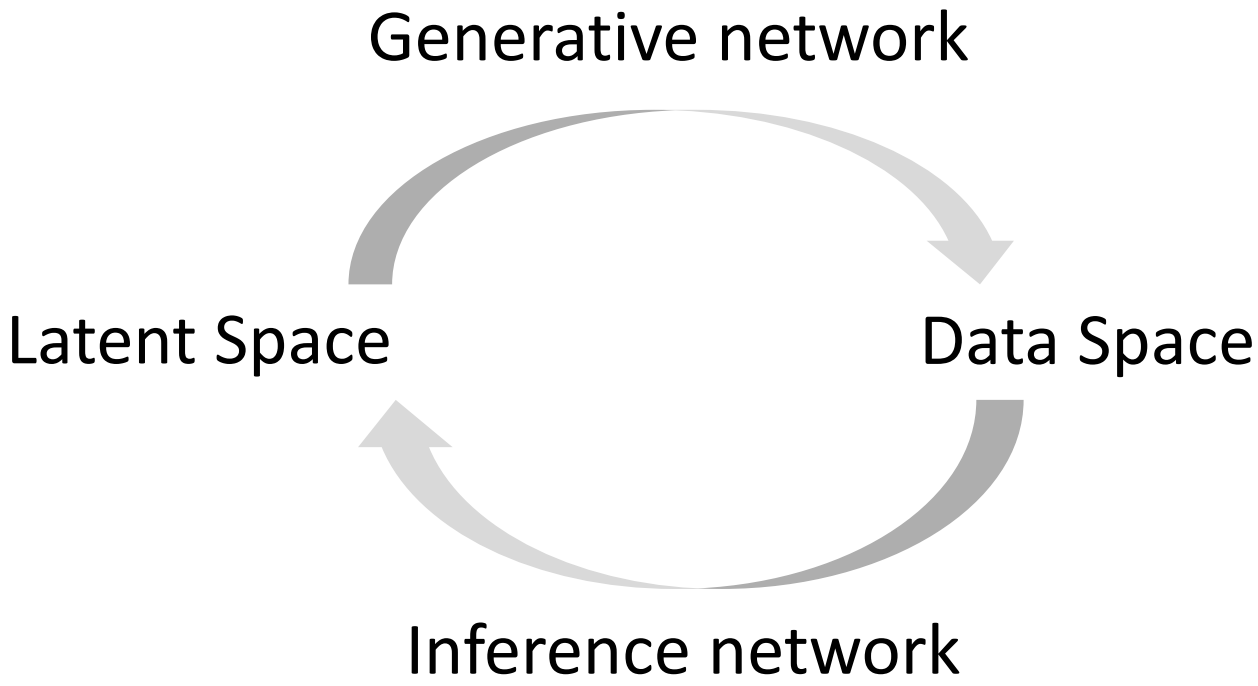
Martin Arjovsky[*]
martinarjovsky@gmail.com

Olivier Mastropietro[*]
oli.mastro@gmail.com

Aaron Courville[*], CIFAR Fellow
aaron.courville@gmail.com

Abstract

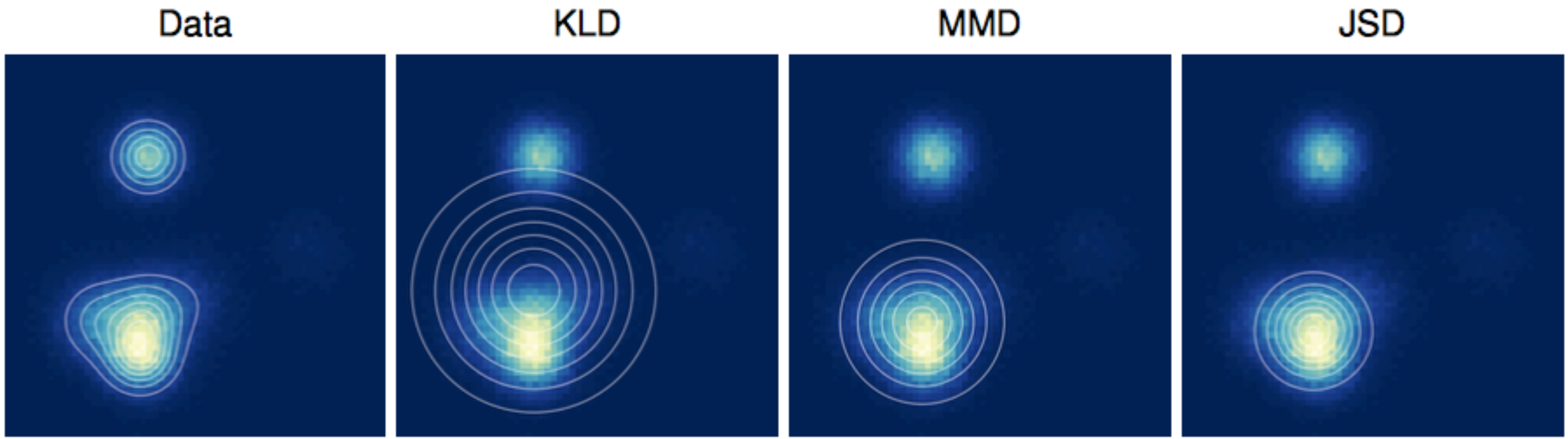
We introduce the adversarially learned inference (ALI) model, which jointly learns a generation network and an inference network using an adversarial process. The generation network maps samples from stochastic latent variables to the data space while the inference network maps training examples in data space to the space of latent variables. An adversarial game is cast between these two networks and a discriminative network that is trained to distinguish between joint latent/data-space samples from the generative network and joint samples from the inference network. We illustrate the ability of the model to learn mutually coherent inference and generation networks through the inspections of model samples and reconstructions and confirm the usefulness of the learned representations by obtaining a performance competitive with other recent approaches on the semi-supervised SVHN task.

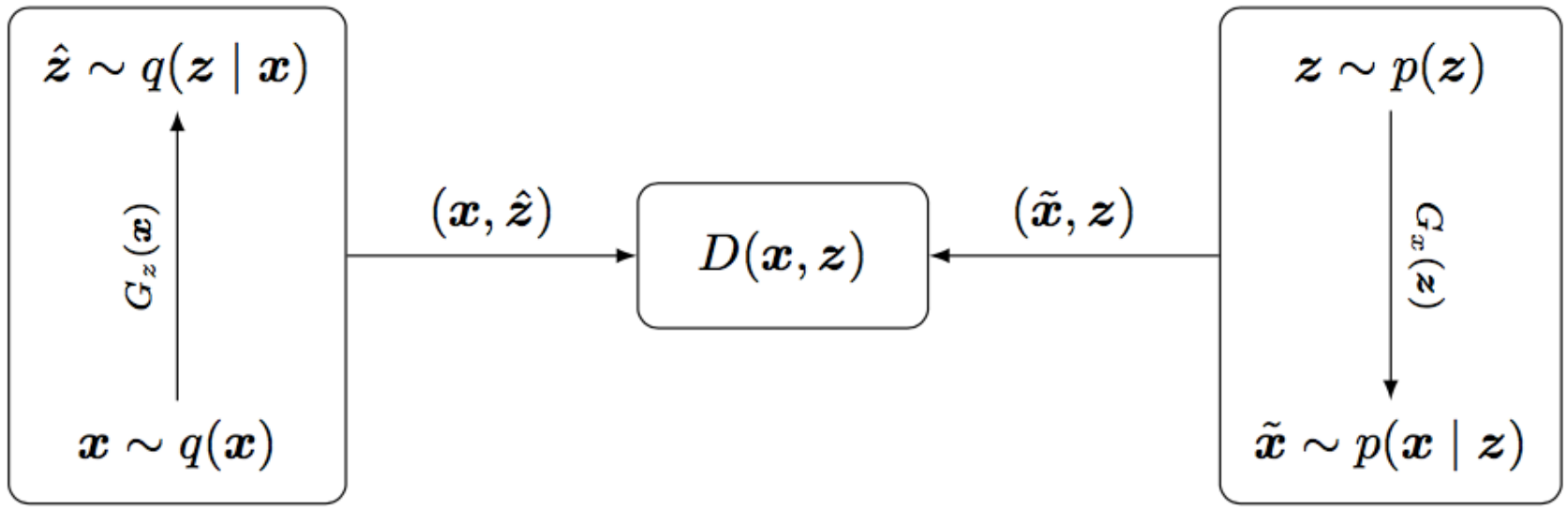

Generative network
Latent Space
Data Space
Inference network

VAE vs. GAN

- VAE
  - Maximizing log-likelihood p(x)
  - Approximate posterior q(z|x)
  - Diffusion problem: All probability mass lying outside the relatively restrictive subset of pixel space occupied by natural images
  - Blurry images

- GAN
  - Hard to extend to semi-supervised learning
  - No efficient inference mechanism


Data
KLD
MMD
JSD



- the *encoder* joint distribution $q(\boldsymbol{x}, \boldsymbol{z}) = q(\boldsymbol{x})q(\boldsymbol{z} \mid \boldsymbol{x})$,
- the *decoder* joint distribution $p(\boldsymbol{x}, \boldsymbol{z}) = p(\boldsymbol{z})p(\boldsymbol{x} \mid \boldsymbol{z})$.

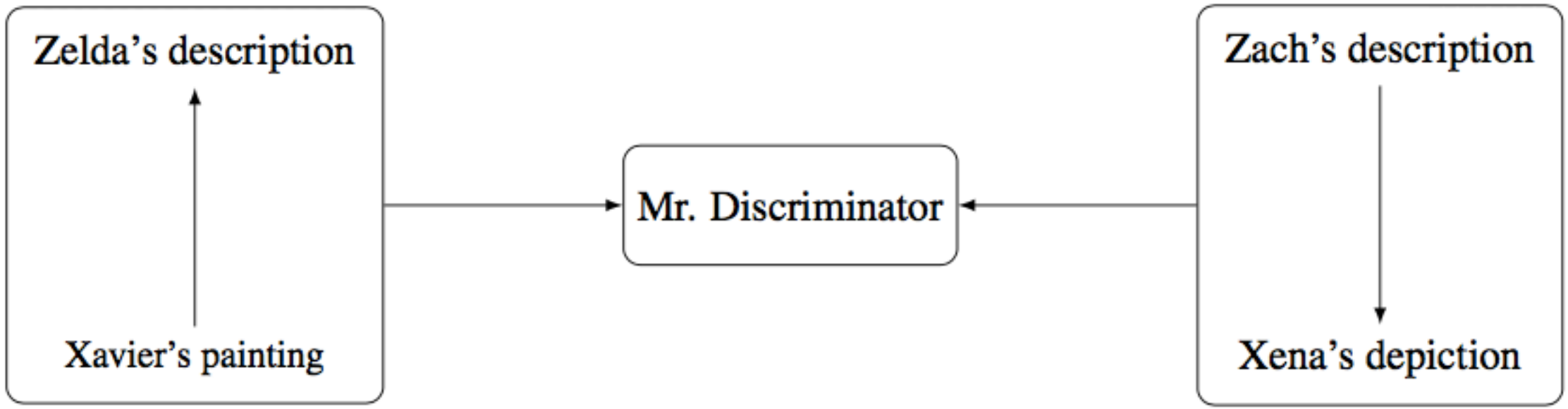


Figure 8: A Circle of Infinite Painters' view of the ALI game.

These two distributions have marginals that are known to us: the encoder marginal $q(\boldsymbol{x})$ is the empirical data distribution and the decoder marginal $p(\boldsymbol{z})$ is usually defined to be a simple, factorized distribution, such as the standard Normal distribution $p(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. As such, the generative process between $q(\boldsymbol{x}, \boldsymbol{z})$ and $p(\boldsymbol{x}, \boldsymbol{z})$ is reversed.

Game Value Function

$$\min_G \max_D V(D,G) = \mathbb{E}_{q(\boldsymbol{x})}[\log(D(\boldsymbol{x}, G_z(\boldsymbol{x})))] + \mathbb{E}_{p(\boldsymbol{z})}[\log(1 - D(G_x(\boldsymbol{z}), \boldsymbol{z}))]$$

$$= \iint q(\boldsymbol{x})q(\boldsymbol{z} \mid \boldsymbol{x}) \log(D(\boldsymbol{x}, \boldsymbol{z}))d\boldsymbol{x}d\boldsymbol{z}$$

$$+ \iint p(\boldsymbol{z})p(\boldsymbol{x} \mid \boldsymbol{z}) \log(1 - D(\boldsymbol{x}, \boldsymbol{z}))d\boldsymbol{x}d\boldsymbol{z}.$$

Sampling from $q(\boldsymbol{z} \mid \boldsymbol{x})$ amounts to drawing an $\boldsymbol{x}$ from the data distribution and propagating the sample through the encoder network $(\boldsymbol{\mu}_z, \boldsymbol{\sigma}_z) = G_z(\boldsymbol{x})$, then generating $\hat{\boldsymbol{z}} \sim \mathcal{N}(\boldsymbol{\mu}_z, \boldsymbol{\sigma}_z)$. Sampling from $p(\boldsymbol{x} \mid \boldsymbol{z})$ amounts to sampling from $p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ and propagating the sample through the decoder network to recover $\tilde{\boldsymbol{x}}$.

Generator: minimizing Jenson-Shannon Divergence between $q(\boldsymbol{x}, \boldsymbol{z})$ and $p(\boldsymbol{x}, \boldsymbol{z})$

$$\mathrm{JSD}(P \parallel Q) = \frac{1}{2}D(P \parallel M) + \frac{1}{2}D(Q \parallel M)$$

$$M = \frac{1}{2}(P + Q)$$

$$\mathrm{JSD}_{\pi_1,\ldots,\pi_n}(P_1, P_2, \ldots, P_n) = H\left(\sum_{i=1}^{n} \pi_i P_i\right) - \sum_{i=1}^{n} \pi_i H(P_i)$$

Compared with GAN

- The generator has two components: the encoder, $G_z(\boldsymbol{x})$, which maps data samples $\boldsymbol{x}$ to $\boldsymbol{z}$-space, and the decoder $G_x(\boldsymbol{z})$, which maps samples from the prior $p(\boldsymbol{z})$ (a source of

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

- The discriminator is trained to distinguish between joint pairs $(\boldsymbol{x}, \hat{\boldsymbol{z}} = G_x(\boldsymbol{x}))$ and $(\tilde{\boldsymbol{x}} = G_x(\boldsymbol{z}), \boldsymbol{z})$, as opposed to marginal samples $\boldsymbol{x} \sim q(\boldsymbol{x})$ and $\tilde{\boldsymbol{x}} \sim p(\boldsymbol{x})$.

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

Reparametrization Trick (VAE)

$$q(z \mid x) = \mathcal{N}(\mu(x), \sigma^2(x)I)$$

$$z = \mu(x) + \sigma(x) \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

Maximizing, preventing zero gradient

$$V'(D, G) = \mathbb{E}_{q(\boldsymbol{x})}[\log(1 - D(\boldsymbol{x}, G_{\boldsymbol{z}}(\boldsymbol{x})))] + \mathbb{E}_{p(\boldsymbol{z})}[\log(D(G_{\boldsymbol{x}}(\boldsymbol{z}), \boldsymbol{z}))]$$

$$D^*(x, z) = \frac{q(x, z)}{q(x, z) + p(x, z)}.$$

Algorithm 1 The ALI training procedure.

$\theta_g, \theta_d \leftarrow$ initialize network parameters

repeat

$\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(M)} \sim q(\boldsymbol{x})$ ▷ Draw M samples from the dataset and the prior

$\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(M)} \sim p(\boldsymbol{z})$

$\hat{\boldsymbol{z}}^{(i)} \sim q(\boldsymbol{z} \mid \boldsymbol{x} = \boldsymbol{x}^{(i)}), \quad i = 1, \ldots, M$ ▷ Sample from the conditionals

$\tilde{\boldsymbol{x}}^{(j)} \sim p(\boldsymbol{x} \mid \boldsymbol{z} = \boldsymbol{z}^{(j)}), \quad j = 1, \ldots, M$

$\rho_q^{(i)} \leftarrow D(\boldsymbol{x}^{(i)}, \hat{\boldsymbol{z}}^{(i)}), \quad i = 1, \ldots, M$ ▷ Compute discriminator predictions

$\rho_p^{(j)} \leftarrow D(\tilde{\boldsymbol{x}}^{(j)}, \boldsymbol{z}^{(j)}), \quad j = 1, \ldots, M$

$\mathcal{L}_d \leftarrow -\frac{1}{M} \sum_{i=1}^{M} \log(\rho_q^{(i)}) - \frac{1}{M} \sum_{j=1}^{M} log(1 - \rho_p^{(j)})$ ▷ Compute discriminator loss

$\mathcal{L}_g \leftarrow -\frac{1}{M} \sum_{i=1}^{M} \log(1 - \rho_q^{(i)}) - \frac{1}{M} \sum_{j=1}^{M} \log(\rho_p^{(j)})$ ▷ Compute generator loss

$\theta_d \leftarrow \theta_d - \nabla_{\theta_d} \mathcal{L}_d$ ▷ Gradient update on discriminator network

$\theta_g \leftarrow \theta_g - \nabla_{\theta_g} \mathcal{L}_g$ ▷ Gradient update on generator networks

until convergence

Latent Space Interpolation

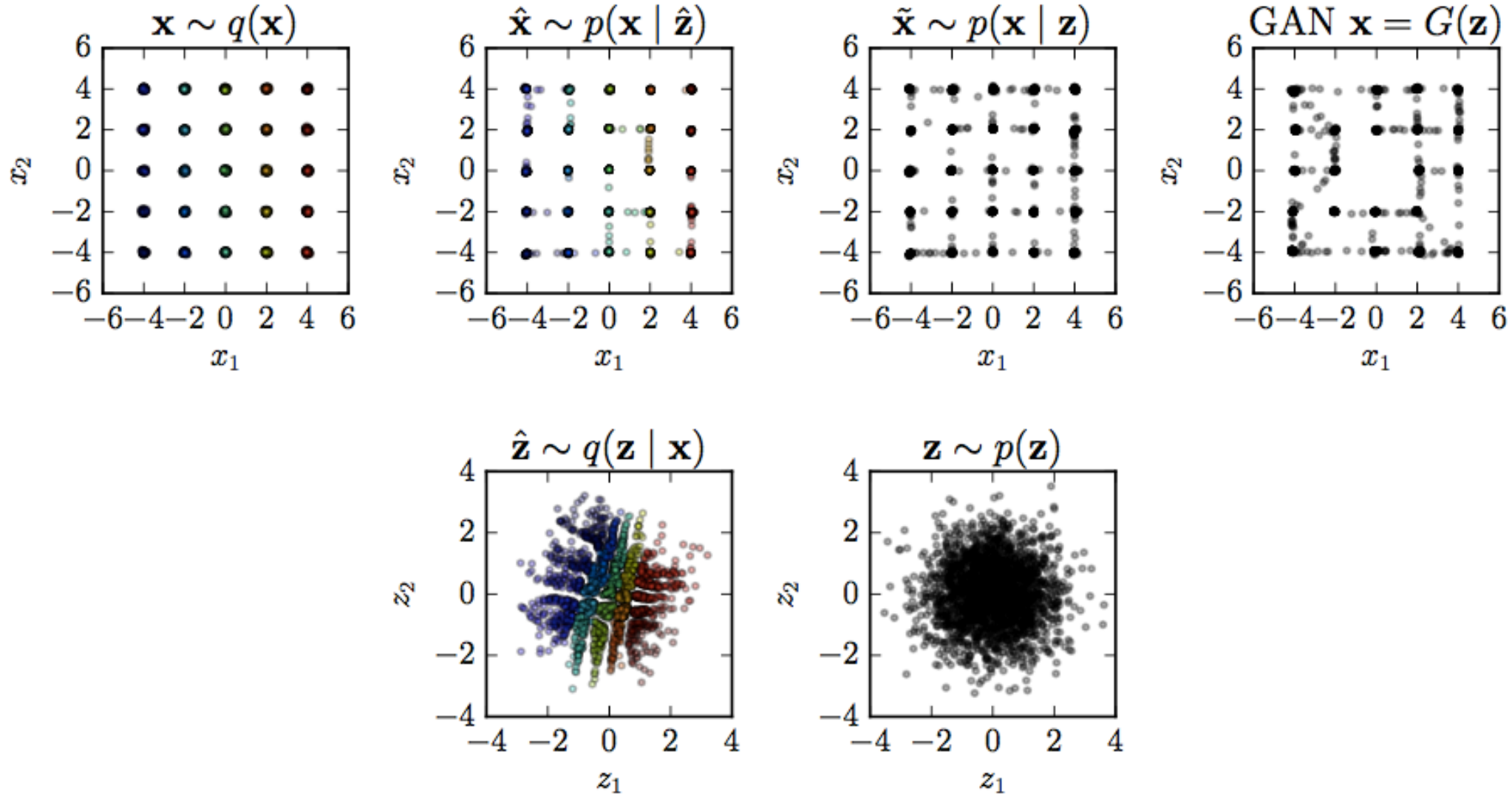


Figure 7: Comparison of ALI and GAN on a synthetic gaussian mixture dataset. In the top row, from left to right: data samples, ALI reconstructions, ALI samples, GAN samples. In the bottom row, from left to right: ALI encodings, prior samples. The nine mixture modes, their ALI encodings and their ALI reconstructions have been color-coded to facilitate visualization.

Semi-supervised

We follow the procedure outlined by Radford *et al.* (2015). We train an L2-SVM on the learned representations of a model trained on SVHN. The last three hidden layers of the encoder as well as its output are concatenated to form a 8960-dimensional feature vector. A 10,000 example held-out validation set is taken from the training set and is used for model selection. The SVM is trained on 1000 examples taken at random from the remainder of the training set. The test error rate is measured for 100 different SVMs trained on different random 1000-example training sets, and the average error rate is measured along with its standard deviation.

Method	Error rate
KNN (as reported in Zhao *et al.* (2015))	77.93%
TSVM (Vapnik, 1998)	66.55%
VAE (M1 + M2) (Kingma *et al.*, 2014)	36.02%
SWWAE without dropout (Zhao *et al.*, 2015)	27.83%
SWWAE with dropout (Zhao *et al.*, 2015)	23.56%
DCGAN + L2-SVM (Radford *et al.*, 2015)	22.18%($\pm$1.13%)
SDGM (Maaløe *et al.*, 2016)	**16.61%($\pm$0.24%)**
ALI (ours)	19.14%($\pm$0.50%)

Table 1: SVHN test set error rates for semi-supervised learning. 1000 training examples were used.

Operation	Kernel	Strides	Feature maps	BN?	Dropout	Nonlinearity
$G_x(x)$ – $3 \times 32 \times 32$ input						
Convolution	5×5	1×1	32	$\surd$	0.0	Leaky ReLU
Convolution	4×4	2×2	64	$\surd$	0.0	Leaky ReLU
Convolution	4×4	1×1	128	$\surd$	0.0	Leaky ReLU
Convolution	4×4	2×2	256	$\surd$	0.0	Leaky ReLU
Convolution	4×4	1×1	512	$\surd$	0.0	Leaky ReLU
Convolution	1×1	1×1	512	$\surd$	0.0	Leaky ReLU
Convolution	1×1	1×1	512	$\times$	0.0	Linear
$G_z(z)$ – $256 \times 1 \times 1$ input						
Transposed convolution	4×4	1×1	256	$\surd$	0.0	Leaky ReLU
Transposed convolution	4×4	2×2	128	$\surd$	0.0	Leaky ReLU
Transposed convolution	4×4	1×1	64	$\surd$	0.0	Leaky ReLU
Transposed convolution	4×4	2×2	32	$\surd$	0.0	Leaky ReLU
Transposed convolution	5×5	1×1	32	$\surd$	0.0	Leaky ReLU
Convolution	1×1	1×1	32	$\surd$	0.0	Leaky ReLU
Convolution	1×1	1×1	3	$\times$	0.0	Sigmoid
$D(x)$ – $3 \times 32 \times 32$ input						
Convolution	5×5	1×1	32	$\times$	0.2	Leaky ReLU
Convolution	4×4	2×2	64	$\surd$	0.2	Leaky ReLU
Convolution	4×4	1×1	128	$\surd$	0.2	Leaky ReLU
Convolution	4×4	2×2	256	$\surd$	0.2	Leaky ReLU
Convolution	4×4	1×1	512	$\surd$	0.2	Leaky ReLU
$D(z)$ – $256 \times 1 \times 1$ input						
Convolution	1×1	1×1	512	$\times$	0.2	Leaky ReLU
Convolution	1×1	1×1	512	$\times$	0.2	Leaky ReLU
$D(x, z)$ – $1024 \times 1 \times 1$ input						
Concatenate $D(x)$ and $D(z)$ along the channel axis						
Convolution	1×1	1×1	1024	$\times$	0.2	Leaky ReLU
Convolution	1×1	1×1	1024	$\times$	0.2	Leaky ReLU
Convolution	1×1	1×1	1	$\times$	0.2	Sigmoid
Optimizer	Adam ($\alpha = 10^{-4}$, $\beta_1 = 0.5$, $\beta_2 = 10^{-3}$)					
Batch size	100					
Epochs	100					
Leaky ReLU slope	0.01					
Weight, bias initialization	Isotropic gaussian ($\mu = 0$, $\sigma = 0.01$), Constant(0)					

Table 3: SVHN model hyperparameters.

Adversarial Feature Learning

Jeff Donahue, Philipp Krähenbühl, Trevor Darrell
Computer Science Division
University of California, Berkeley
{jdonahue,philkr,trevor}@cs.berkeley.edu

Abstract

The ability of the Generative Adversarial Networks (GANs) framework to learn generative models mapping from simple latent distributions to arbitrarily complex data distributions has been demonstrated empirically, with compelling results showing generators learn to "linearize semantics" in the latent space of such models. Intuitively, such latent spaces may serve as useful feature representations for auxiliary problems where semantics are relevant. However, in their existing form, GANs have no means of learning the inverse mapping – projecting data back into the latent space. We propose Bidirectional Generative Adversarial Networks (BiGANs) as a means of learning this inverse mapping, and demonstrate that the resulting learned feature representation is useful for auxiliary supervised discrimination tasks, competitive with contemporary approaches to unsupervised and self-supervised feature learning.

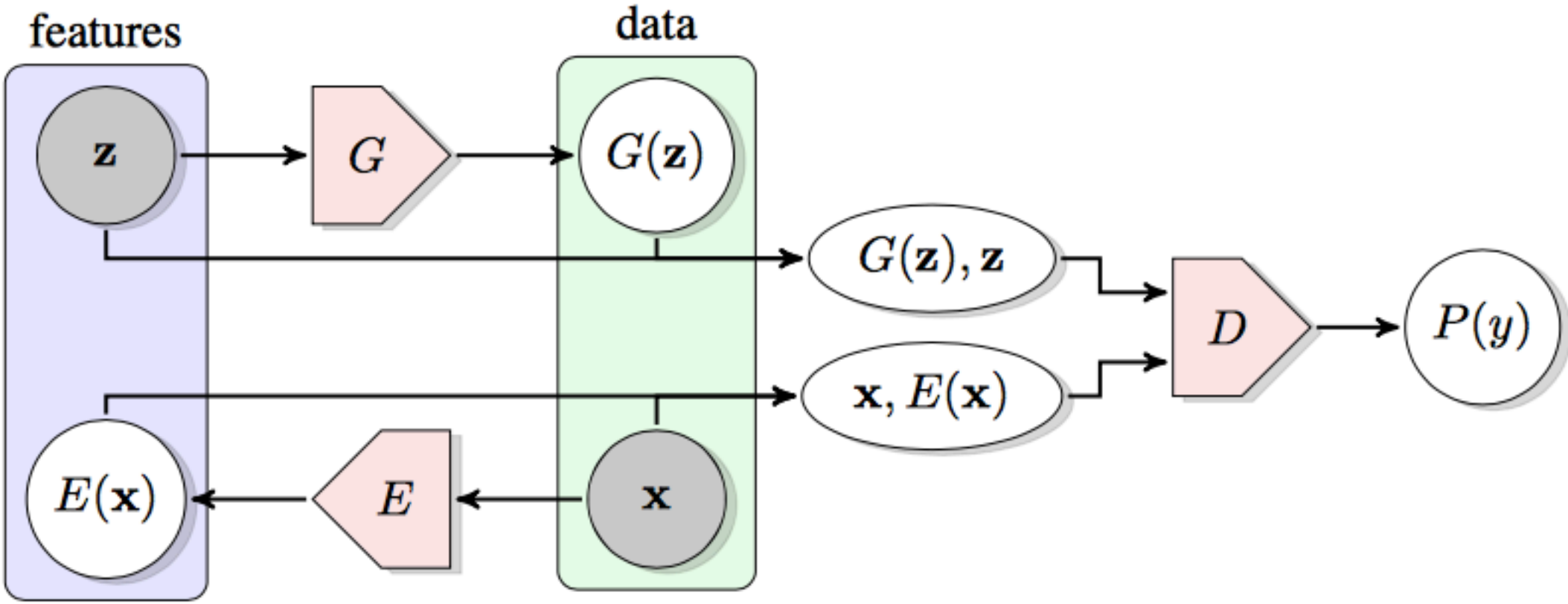


Figure 1: The structure of a Bidirectional Generative Adversarial Network (BiGAN).

$$V(D,E,G) = \mathbb{E}_{\mathbf{x}\sim p\mathbf{x}}\left[\underbrace{\mathbb{E}_{\mathbf{z}\sim p_E(\cdot|\mathbf{x})}\left[\log D(\mathbf{x},\mathbf{z})\right]}_{\log D(\mathbf{x},E(\mathbf{x}))}\right] + \mathbb{E}_{\mathbf{z}\sim p\mathbf{z}}\left[\underbrace{\mathbb{E}_{\mathbf{x}\sim p_G(\cdot|\mathbf{z})}\left[\log\left(1-D(\mathbf{x},\mathbf{z})\right)\right]}_{\log(1-D(G(\mathbf{z}),\mathbf{z}))}\right]$$

- Same value function
- Swapped labels
- Jenson-Shannon

Learning What and Where to Draw

Scott Reed[1,*] reedscot@google.com
Zeynep Akata[2] akata@mpi-inf.mpg.de
Santosh Mohan[1] santoshm@umich.edu
Samuel Tenka[1] samtenka@umich.edu
Bernt Schiele[2] schiele@mpi-inf.mpg.de
Honglak Lee[1] honglak@umich.edu

[1]University of Michigan, Ann Arbor, USA
[2]Max Planck Institute for Informatics, Saarbrücken, Germany

Abstract

Generative Adversarial Networks (GANs) have recently demonstrated the capability to synthesize compelling real-world images, such as room interiors, album covers, manga, faces, birds, and flowers. While existing models can synthesize images based on global constraints such as a class label or caption, they do not provide control over pose or object location. We propose a new model, the Generative Adversarial What-Where Network (GAWWN), that synthesizes images given instructions describing what content to draw in which location. We show high-quality 128×128 image synthesis on the Caltech-UCSD Birds dataset, conditioned on both informal text descriptions and also object location. Our system exposes control over both the bounding box around the bird and its constituent parts. By modeling the conditional distributions over part locations, our system also enables conditioning on arbitrary subsets of parts (e.g. only the beak and tail), yielding an efficient interface for picking part locations. We also show preliminary results on the more challenging domain of text- and location-controllable synthesis of images of human actions on the MPII Human Pose dataset.

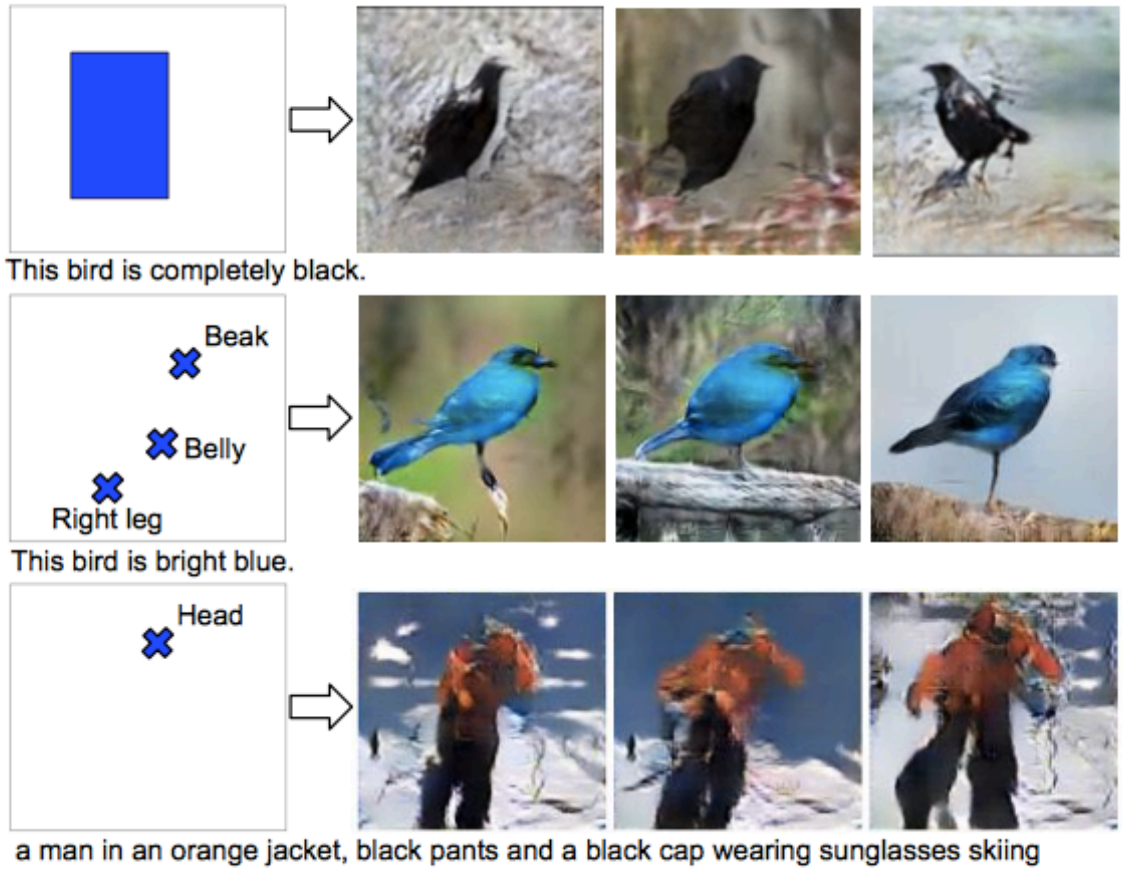


Figure 1: Text-to-image examples. Locations can be specified by keypoint or bounding box.

Text- and location-controllable image synthesis

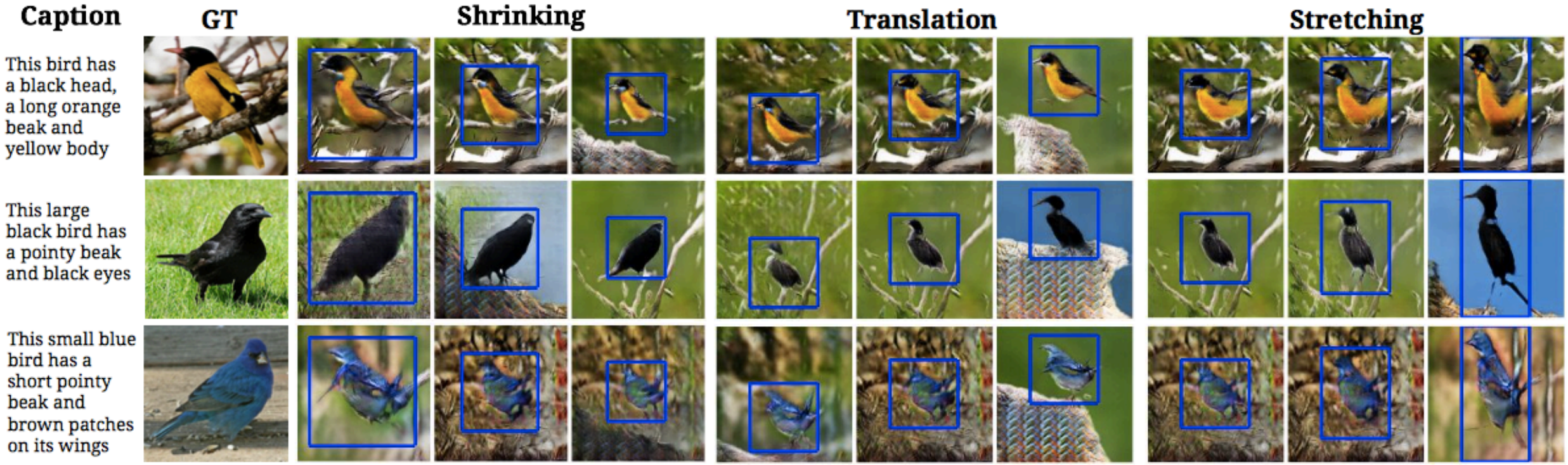


Figure 4: Controlling the bird's position using bounding box coordinates. and previously-unseen text.

Structured Joint Embedding

- Text & image

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

0-1 loss label image text

$$f_v(v) = \arg\max_{y\in\mathcal{Y}} \mathbb{E}_{t\sim\mathcal{T}(y)}[\phi(v)^T\varphi(t))], \quad f_t(t) = \arg\max_{y\in\mathcal{Y}} \mathbb{E}_{v\sim\mathcal{V}(y)}[\phi(v)^T\varphi(t))]$$

image encoder text encoder

training

Generative Adversarial What-Where Networks (GAWWN)

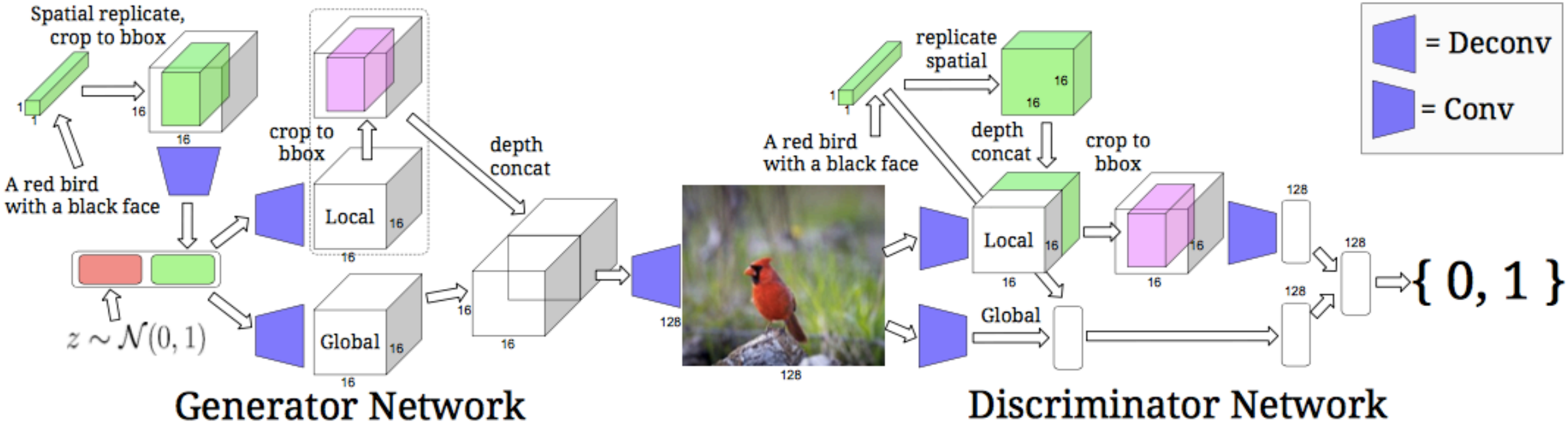


Figure 2: GAWWN with bounding box location control.

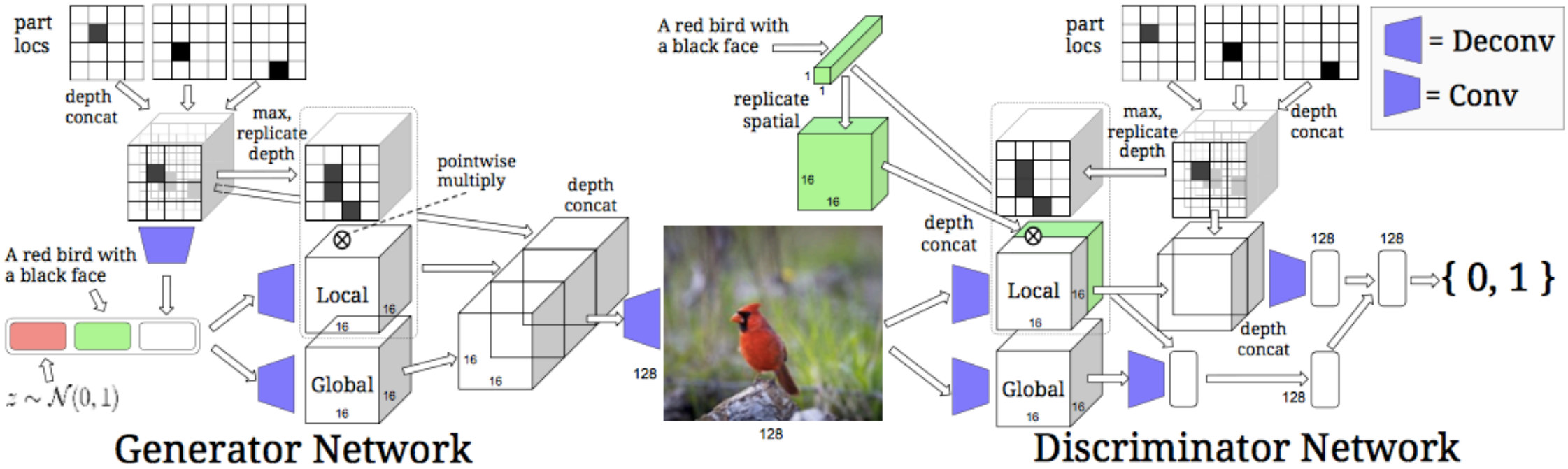


Figure 3: Text and keypoint-conditional GAWWN.. Keypoint grids are shown as 4×4 for clarity of presentation, but in our experiments we used 16×16.

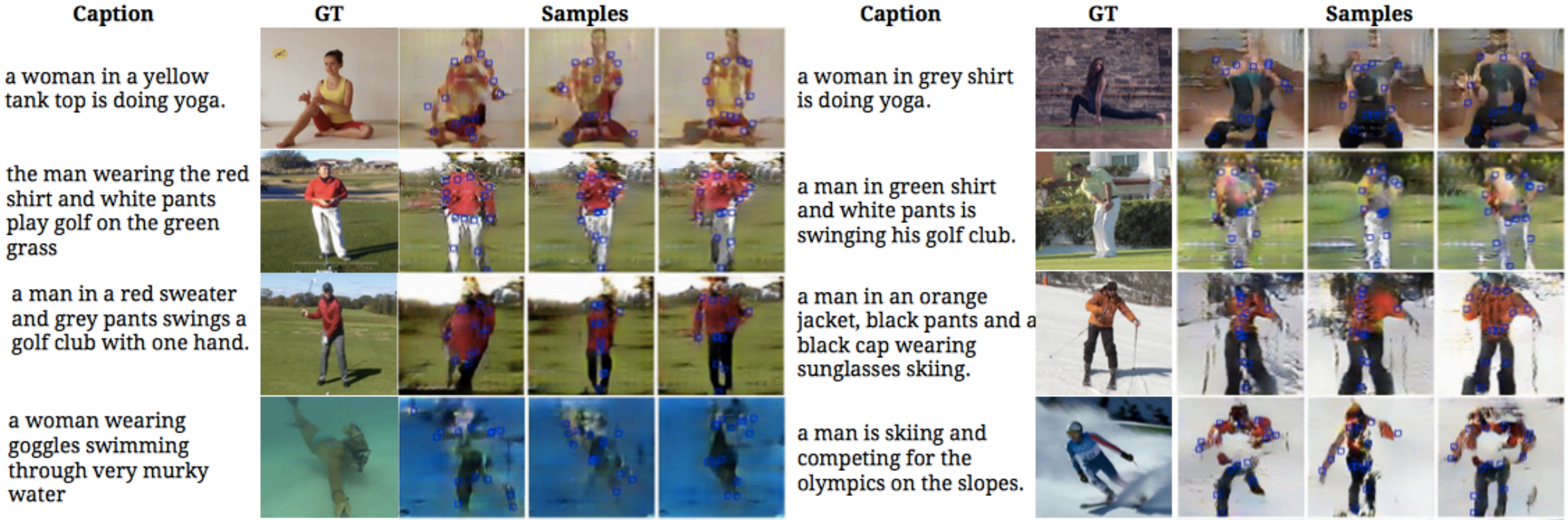


Figure 9: Generating humans. Both the keypoints and the image are generated from unseen text.

Sampling Generative Networks

Notes on a Few Effective Techniques

Tom White
School of Design
Victoria University of Wellington
Wellington, New Zealand
Tom.White@vuw.ac.nz

- Spherical linear Interpolation $$Slerp(q_1, q_2; \boldsymbol{\mu}) = \frac{\sin\ (1 - \mu)\theta}{\sin\theta}\ q_1 + \frac{\sin\mu\theta}{\sin\theta}\ q_2$$

- Analogy:
  - A:B :: C:?
    - ? = C+B-A
    - A:C :: B:?

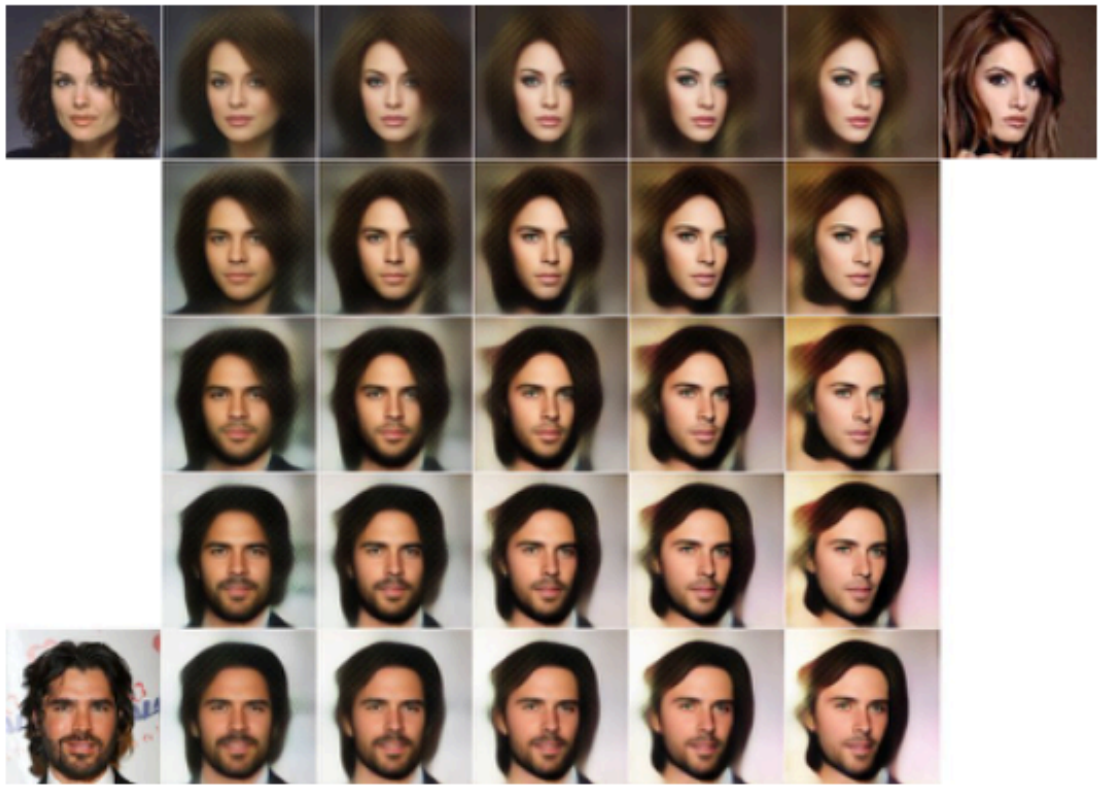


Figure 2: J-Diagram. The three corner images are inputs to the system, with the top left being the "source" (A) and the other two being "analogy targets" (B and C). Adjacent to each is the reconstruction resulting from running the image through both the encoder and decoder of the model. The bottom right image shows the result of applying the analogy operation (B + C) – A. All other images are interpolations using the slerp operator. (model: VAE from Lamb 16 on CelebA)

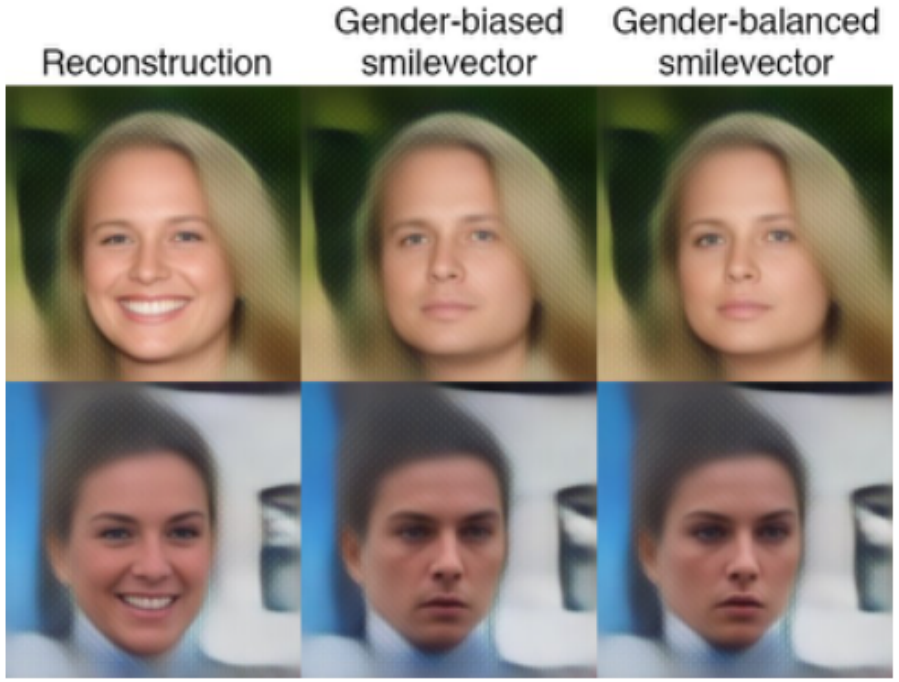
Figure 3: Same J-Diagram repeated with different model type. To facilitate comparisons (and demonstrate results are not cherry-picked) inputs selected are the first 3 images of the validation set. (model: GAN from Dumoulin 16 on CelebA)

MINE—Manifold Interpolation Neighbor Embedding

Latent Space Transformation

- Attribute
  - No smile to smile
- Correlated labels
  - Sample bias

	Male	Not Male	Total
Smile	17%	31%	48%
No Smile	25%	27%	52%
Total	42%	58%	



  - → replicate data

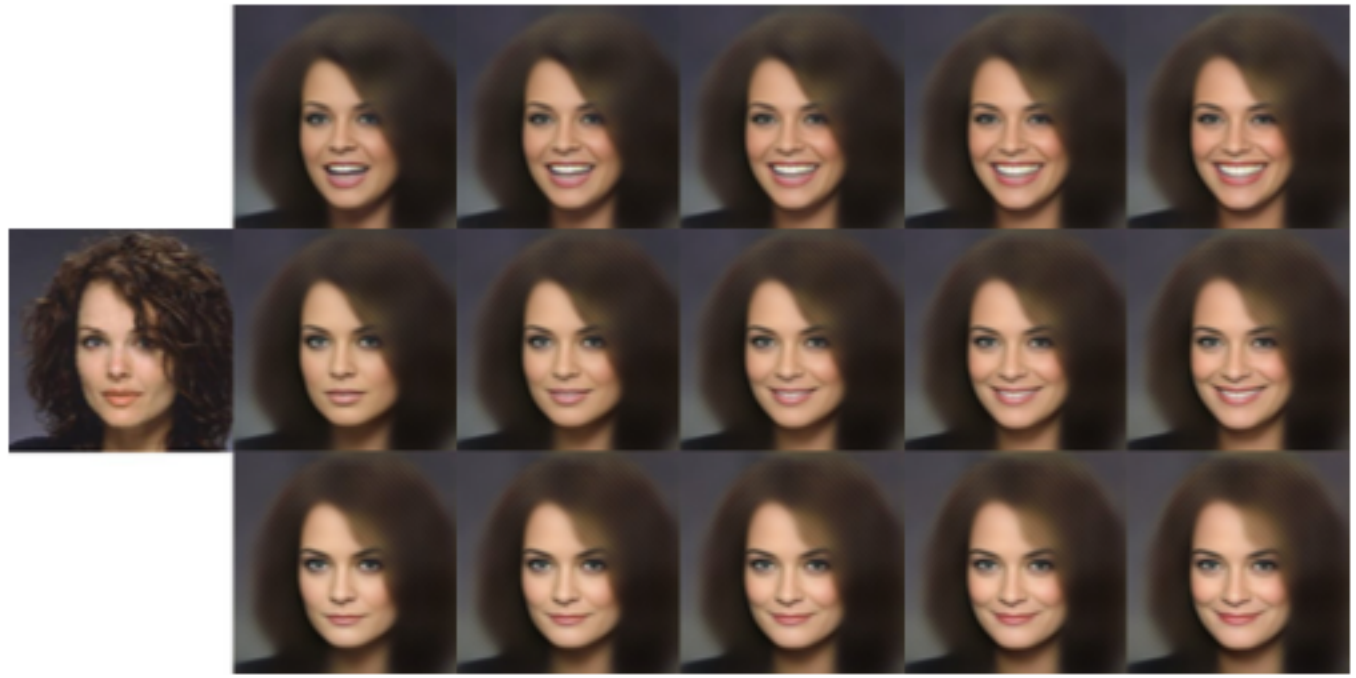


Figure 7: Decoupling attribute vectors for smiling (x-axis) and mouth open (y-axis) allows for more flexible latent space transformations. Input shown at left with reconstruction adjacent. (model: VAE from Lamb 16 on CelebA)

Synthetic Attribute

- Deblurring
  - Extrapolate in the negative direction

Task Specific Adversarial Cost Function

Antonia Creswell, *Student Member, IEEE,* and Anil A. Bharath, *Member, IEEE*

Abstract—The cost function used to train a generative model should fit the purpose of the model. If the model is intended for tasks such as generating perceptually correct samples, it is beneficial to maximise the likelihood of a sample drawn from the model, Q, coming from the same distribution as the training data, P. This is equivalent to minimising the Kullback-Leibler (KL) distance, $KL[Q\|P]$. However, if the model is intended for tasks such as retrieval or classification it is beneficial to maximise the likelihood that a sample drawn from the training data is captured by the model, equivalent to minimising $KL[P\|Q]$. The cost function used in adversarial training optimises the Jensen-Shannon entropy which can be seen as an even interpolation between $KL[Q\|P]$ and $KL[P\|Q]$. Here, we propose an alternative adversarial cost function which allows easy tuning of the model for either task. Our task specific cost function is evaluated on a dataset of hand-written characters in the following tasks: Generation, retrieval and one-shot learning.

The original cost function proposed by Goodfellow et al. [9] is:

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

We propose the alternative cost function:

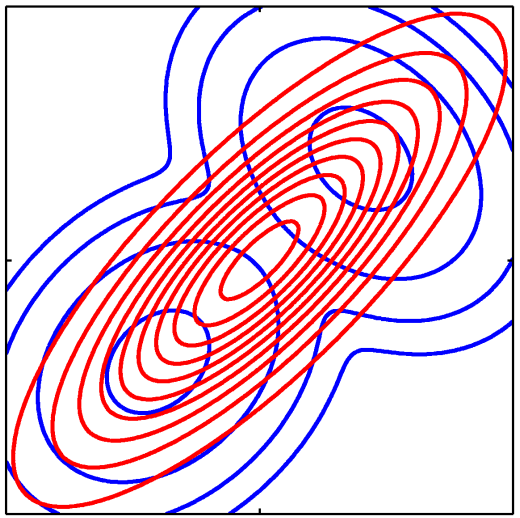
$$\min_G \max_D V(G,D) = \pi \mathbb{E}_{x\sim P} \log D(x) +$$

$$(1-\pi)\mathbb{E}_{z\sim p_z} \log(1 - D(G(z)))$$

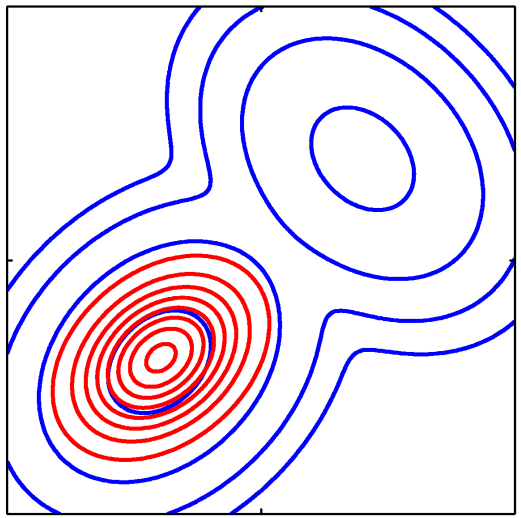
PRML Fig. 10.3 (Bishop)

p: blue

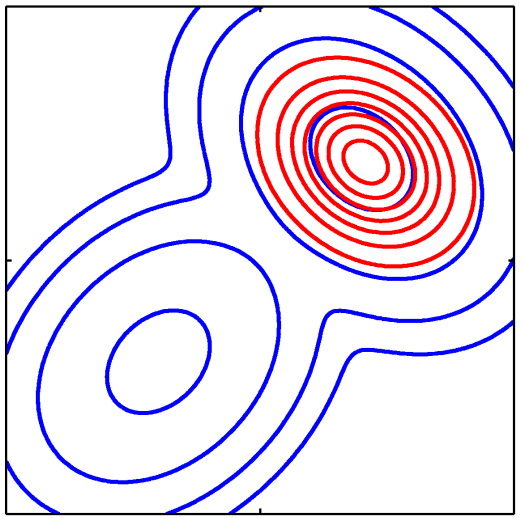
q: red

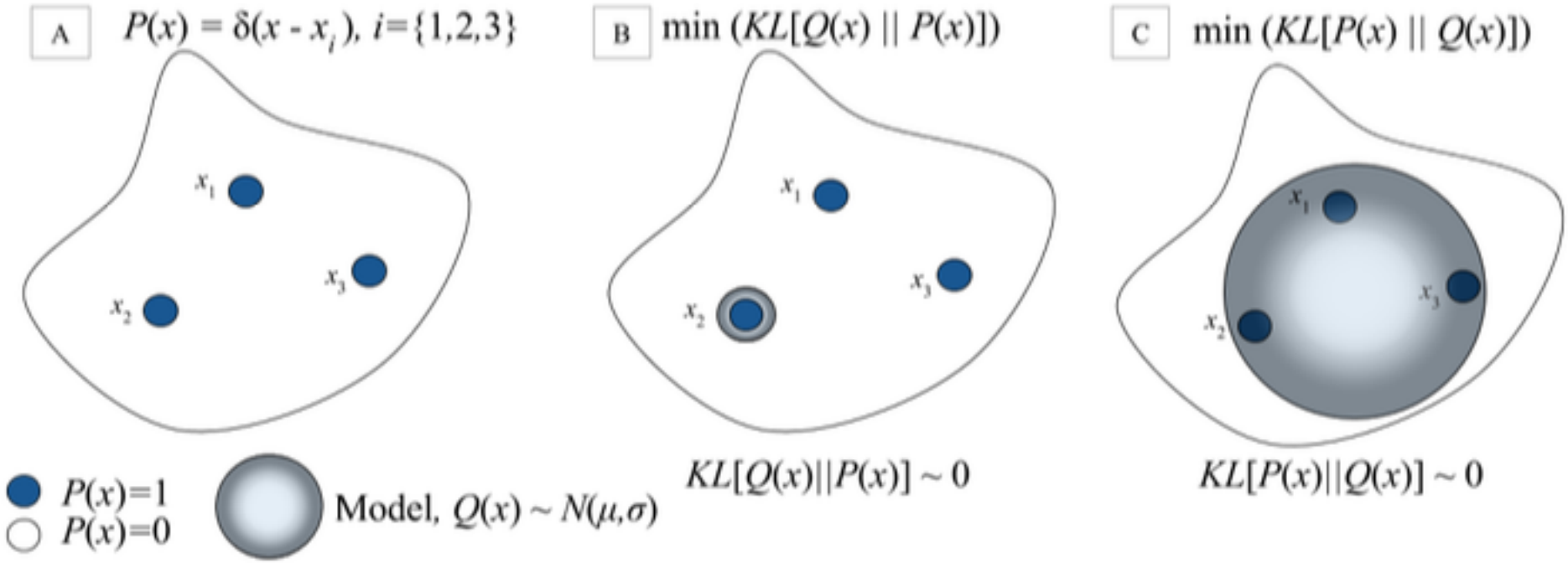


KL(p||q)



KL(q||p)





$$JS_\pi = \pi KL[P\|\pi P+(1-\pi)Q]+(1-\pi)KL[Q\|\pi P+(1-\pi)Q]$$

Huszar et al. [14] showed that JS_π is proportional to $KL[P\|Q]$ and $KL[Q\|P]$, respectively, at the upper and lower limits of π:

$$\frac{JS_\pi[P\|Q]}{\pi} \to KL[P\|Q] \quad \text{as} \quad \pi \to 0$$

$$\frac{JS_\pi[P\|Q]}{(1-\pi)} \to KL[Q\|P] \quad \text{as} \quad \pi \to 1$$