Lecture 15: Unsupervised Deep Learning (III)

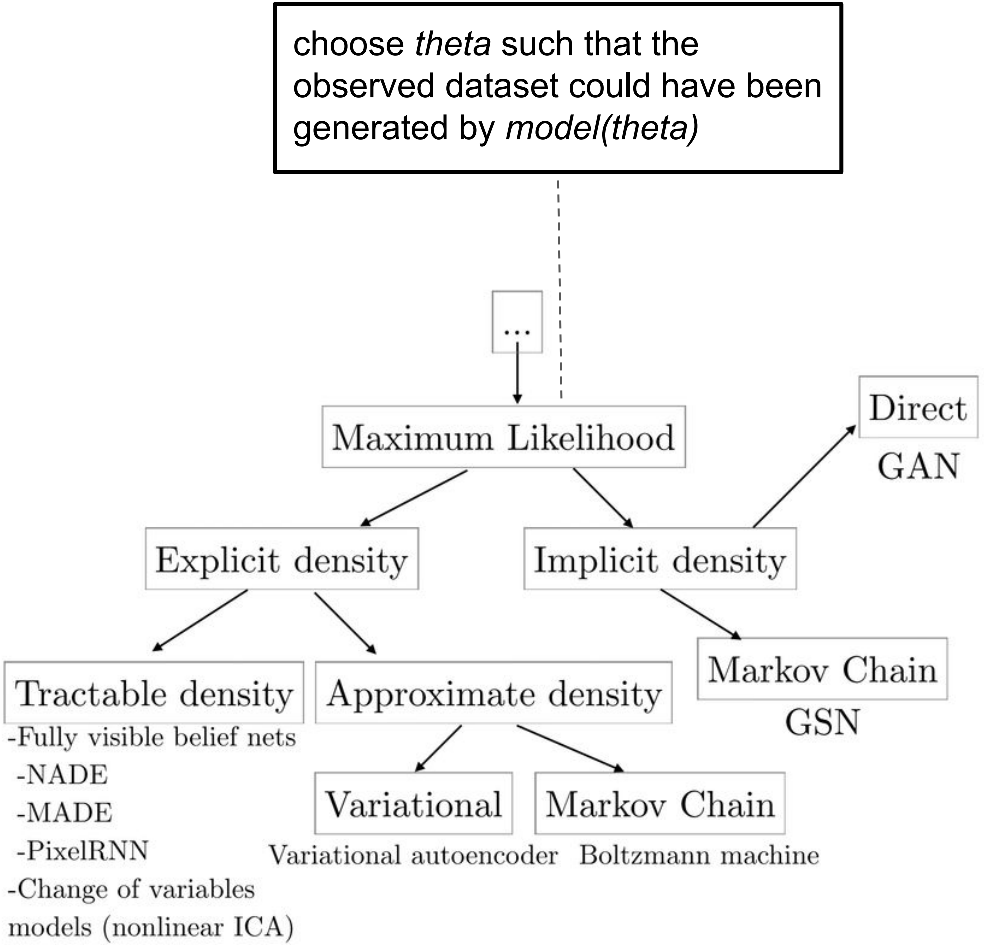
Applications of Generative Models; Normalizing Flows

Outline

1. **Some applications of convolutional autoencoders and GAN**
   a. Image-to-Image Translation with Conditional Adversarial Nets
   b. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks
   c. PuppetGAN: Cross-Domain Image Manipulation by Demonstration

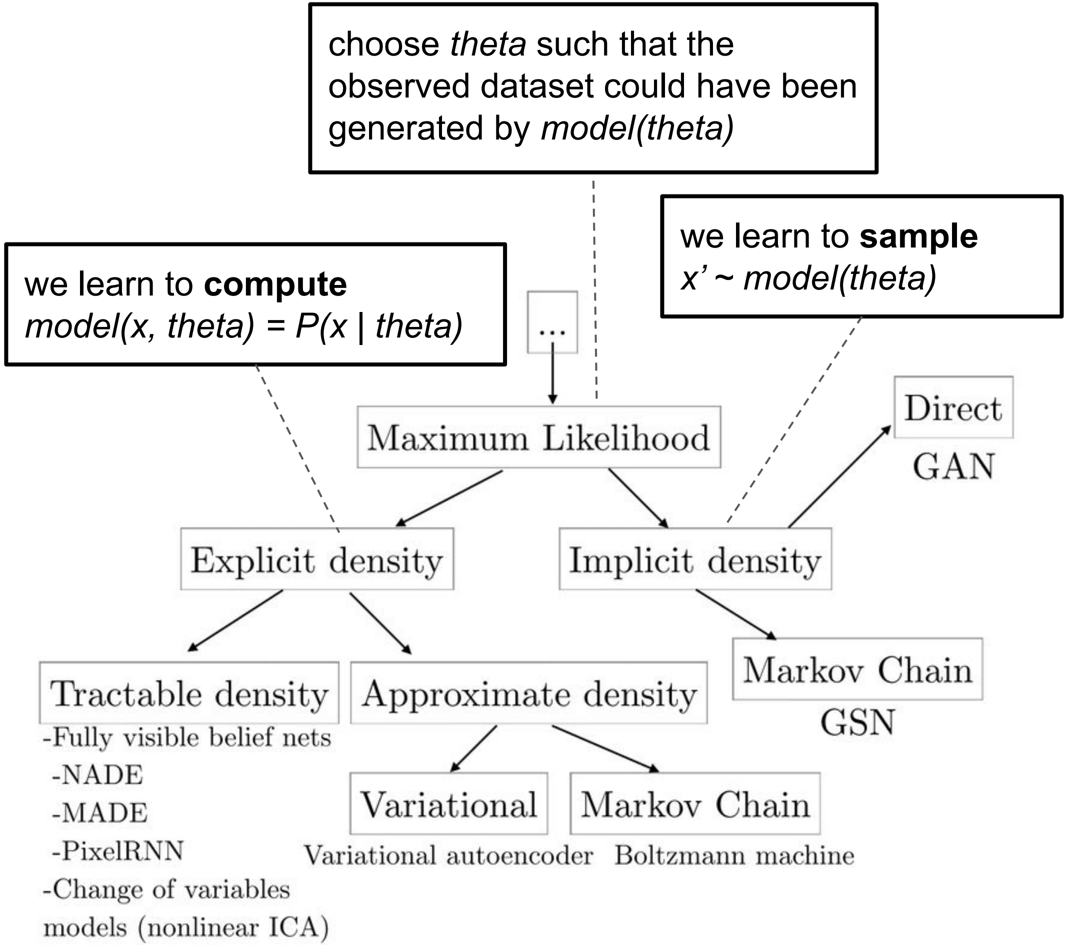
2. Normalizing Flows
   a. Change of variable formula
   b. Planar and radial flows
   c. Real NVP
   d. GLOW
   e. FFJORD
   f. Likelihood vs probability

Recap: GANs



choose *theta* such that the observed dataset could have been generated by *model(theta)*
...
Maximum Likelihood
Explicit density
Implicit density
Direct
GAN
Markov Chain
GSN
Tractable density
-Fully visible belief nets
-NADE
-MADE
-PixelRNN
-Change of variables models (nonlinear ICA)
Approximate density
Variational
Variational autoencoder
Markov Chain
Boltzmann machine

Recap: GANs



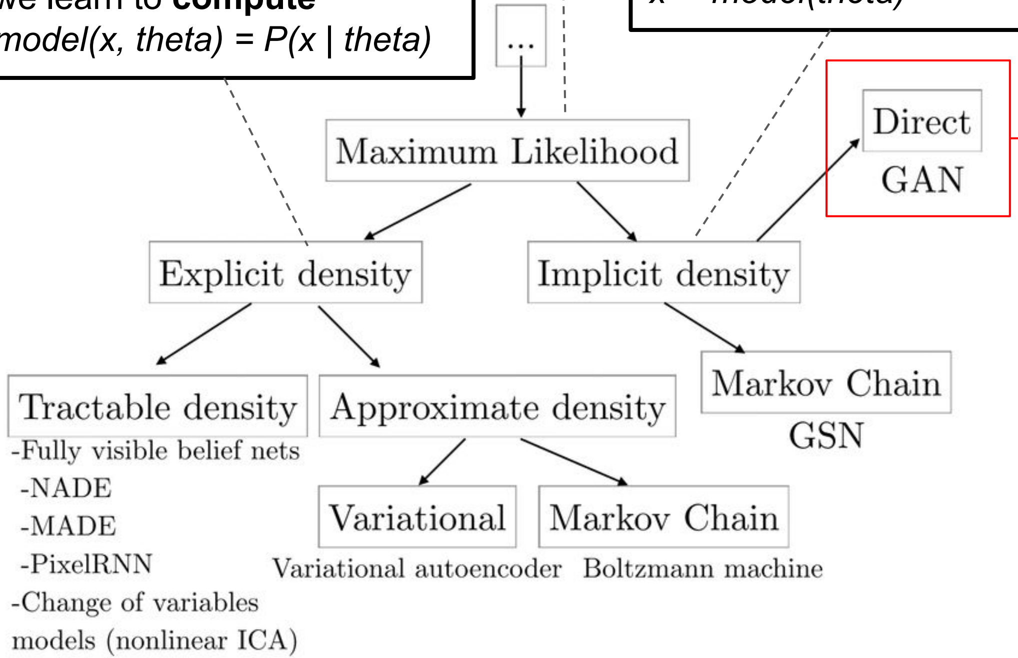
choose *theta* such that the observed dataset could have been generated by *model(theta)*
we learn to **compute** *model(x, theta) = P(x | theta)*
we learn to **sample** *x' ~ model(theta)*
...
Maximum Likelihood
Explicit density
Implicit density
Direct
GAN
Markov Chain
GSN
Tractable density
-Fully visible belief nets
-NADE
-MADE
-PixelRNN
-Change of variables models (nonlinear ICA)
Approximate density
Variational
Variational autoencoder
Markov Chain
Boltzmann machine

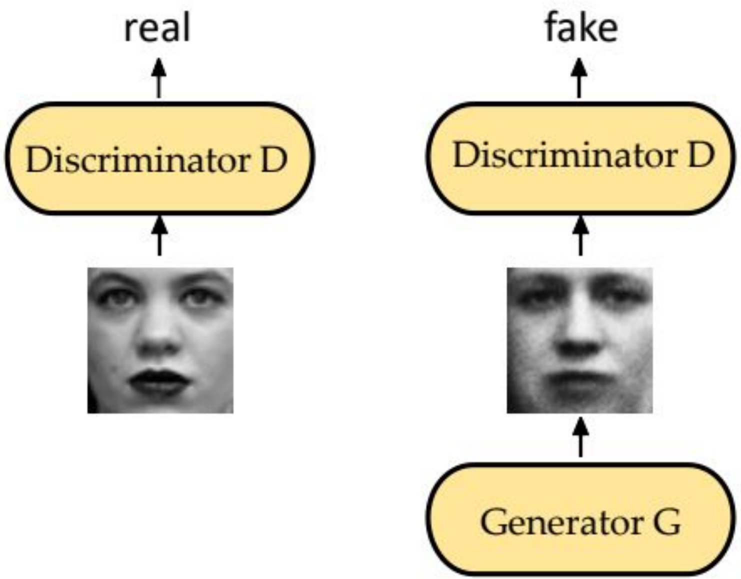
Recap: GANs

choose *theta* such that the observed dataset could have been generated by *model(theta)*

we learn to **compute** *model(x, theta) = P(x | theta)*

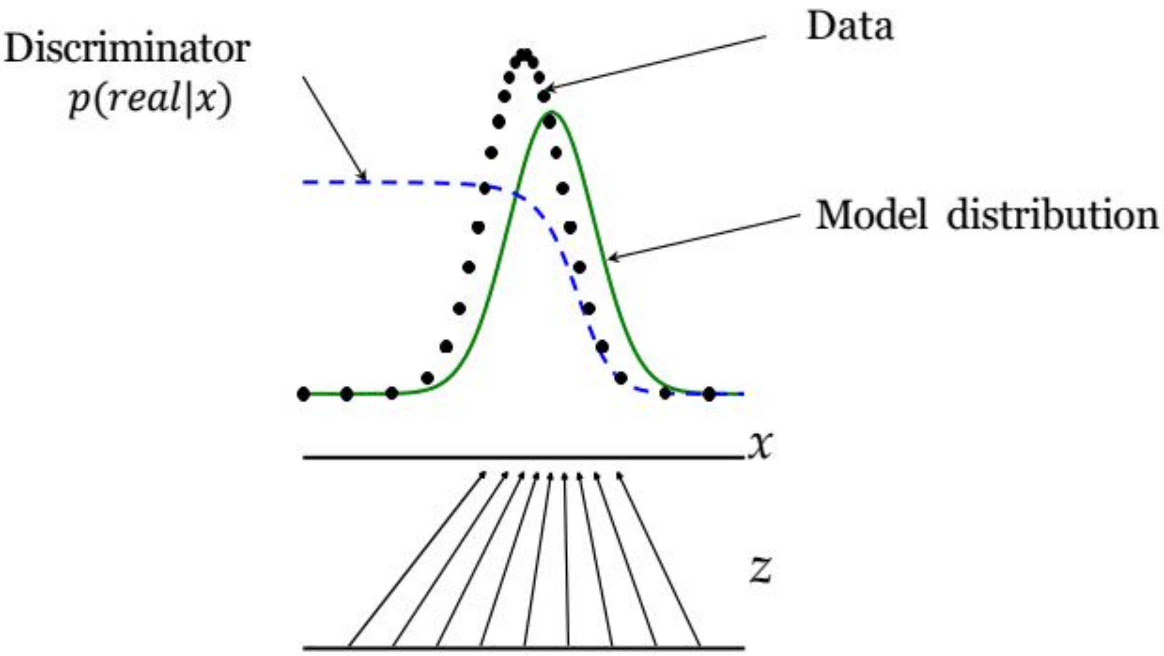
we learn to **sample** *x' ~ model(theta)*



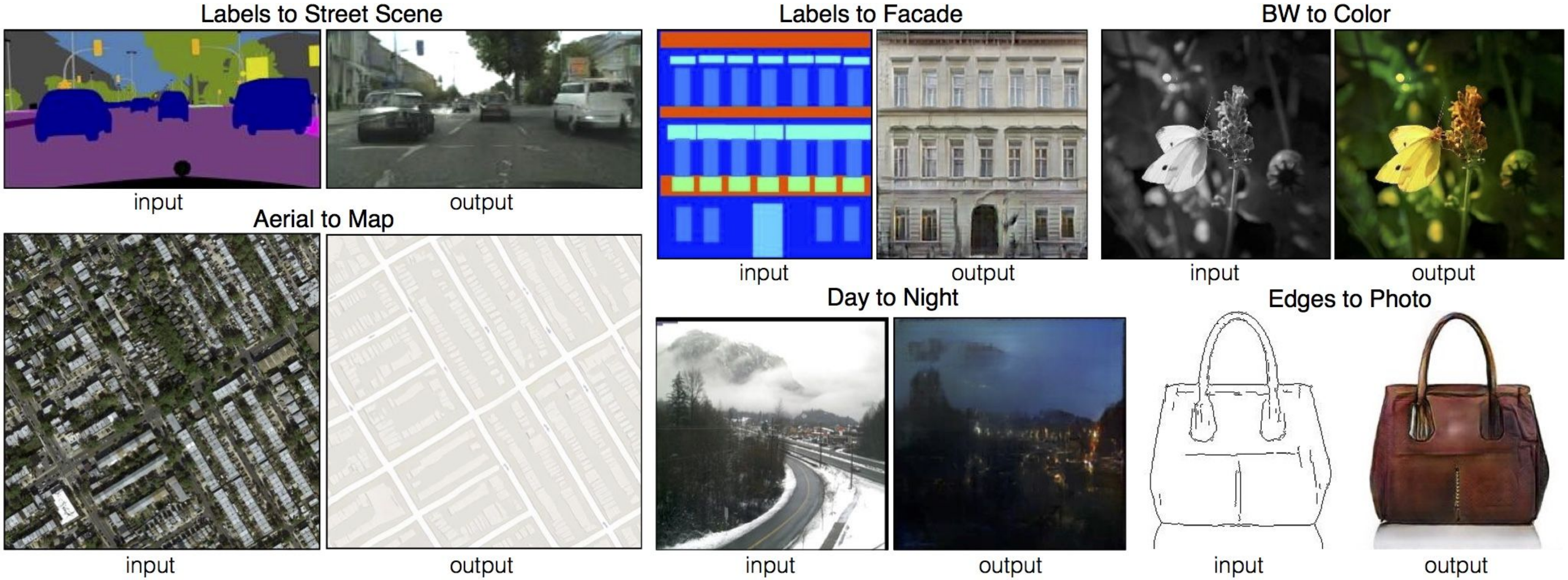


train G “to fool” D,
train D “to catch” G

Recap: GANs

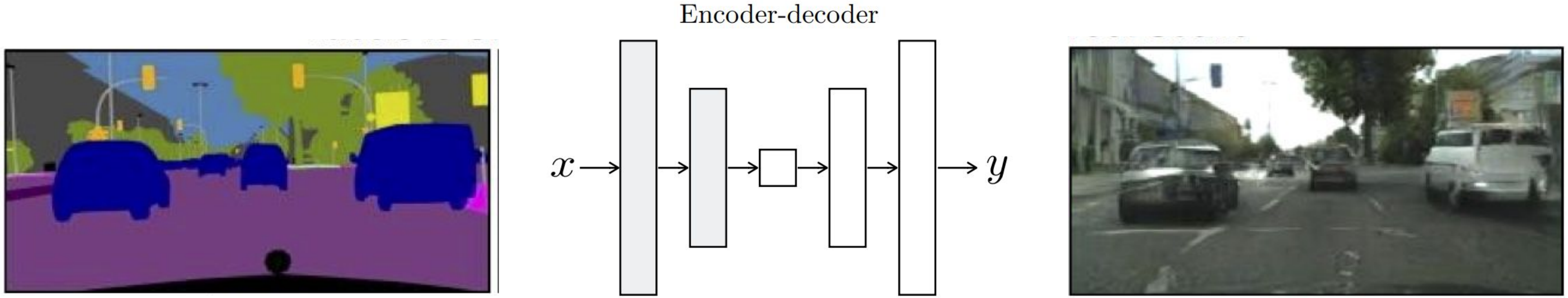


Example Application 1:
Improving outputs of **supervised** image-to-image models



“Image-to-Image Translation with Conditional Adversarial Nets” [Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros]

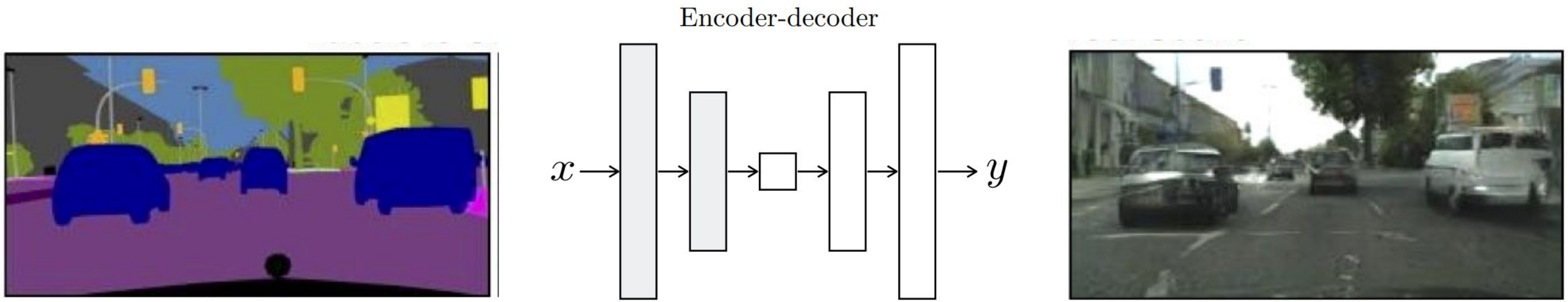
Example Application 1:
Improving outputs of **supervised** image-to-image models



Q: This is a regression model: why not just use a simple supervised loss? (L1, L2)

“Image-to-Image Translation with Conditional Adversarial Nets” [Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros]

Example Application 1:
Improving outputs of **supervised** image-to-image models

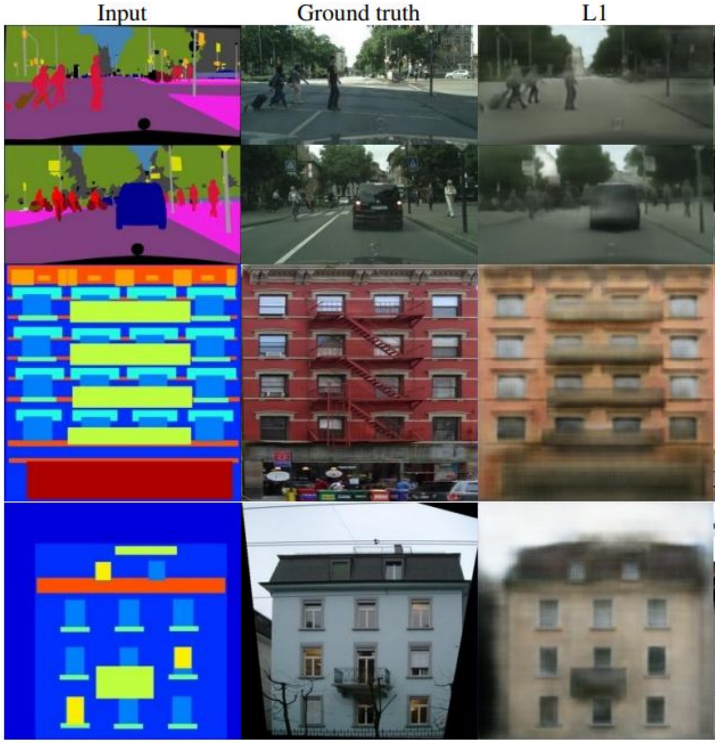


Q: This is a regression model: why not just use a simple supervised loss? (L1, L2)
A: These losses assume that multiple outputs are independent.

$$Y_i = f(X, theta) + e_i \quad e_i \sim \mathrm{N}(0, \mathrm{I}) \quad => \quad \text{L2 loss}$$

“Image-to-Image Translation with Conditional Adversarial Nets” [Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros]

Example Application 1:
Improving outputs of **supervised** image-to-image models



"Image-to-Image Translation with Conditional Adversarial Nets" [Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros]

Example Application 1:
Improving outputs of **supervised** image-to-image models

“Image-to-Image Translation with Conditional Adversarial Nets” [Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros]

Example Application 1:
Improving outputs of **supervised** image-to-image models

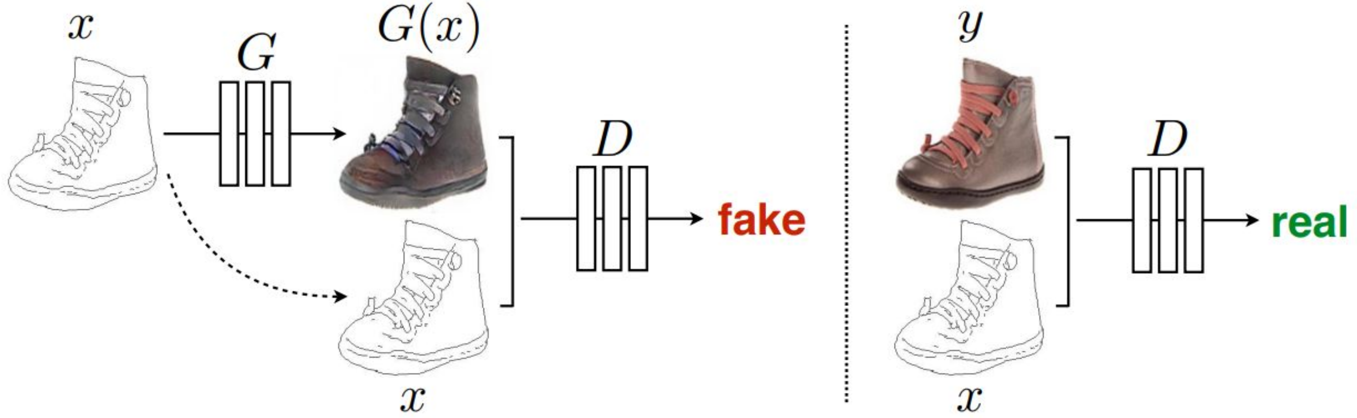


Figure 2: Training a conditional GAN to map edges→photo. The discriminator, D, learns to classify between fake (synthesized by the generator) and real {edge, photo} tuples. The generator, G, learns to fool the discriminator. Unlike an unconditional GAN, both the generator and discriminator observe the input edge map.

Example Application 1:
Improving outputs of **supervised** image-to-image models

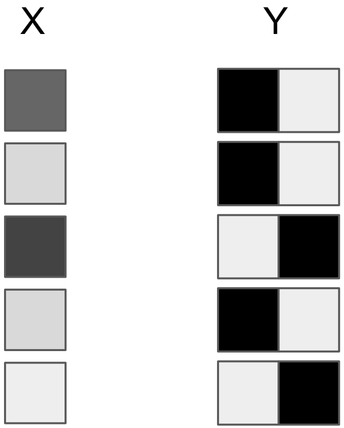
Takeaway:

if the output domain has some structure (i.e. an image)
adversarial losses force the model to follow that structure

Example Application 1:
Improving outputs of **supervised** image-to-image models

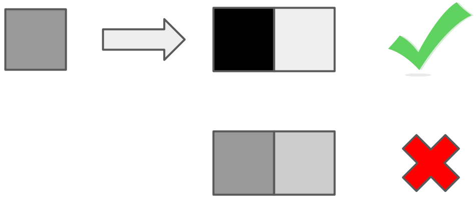
Takeaway:

if the output domain has some structure (i.e. an image)
adversarial losses force the model to follow that structure

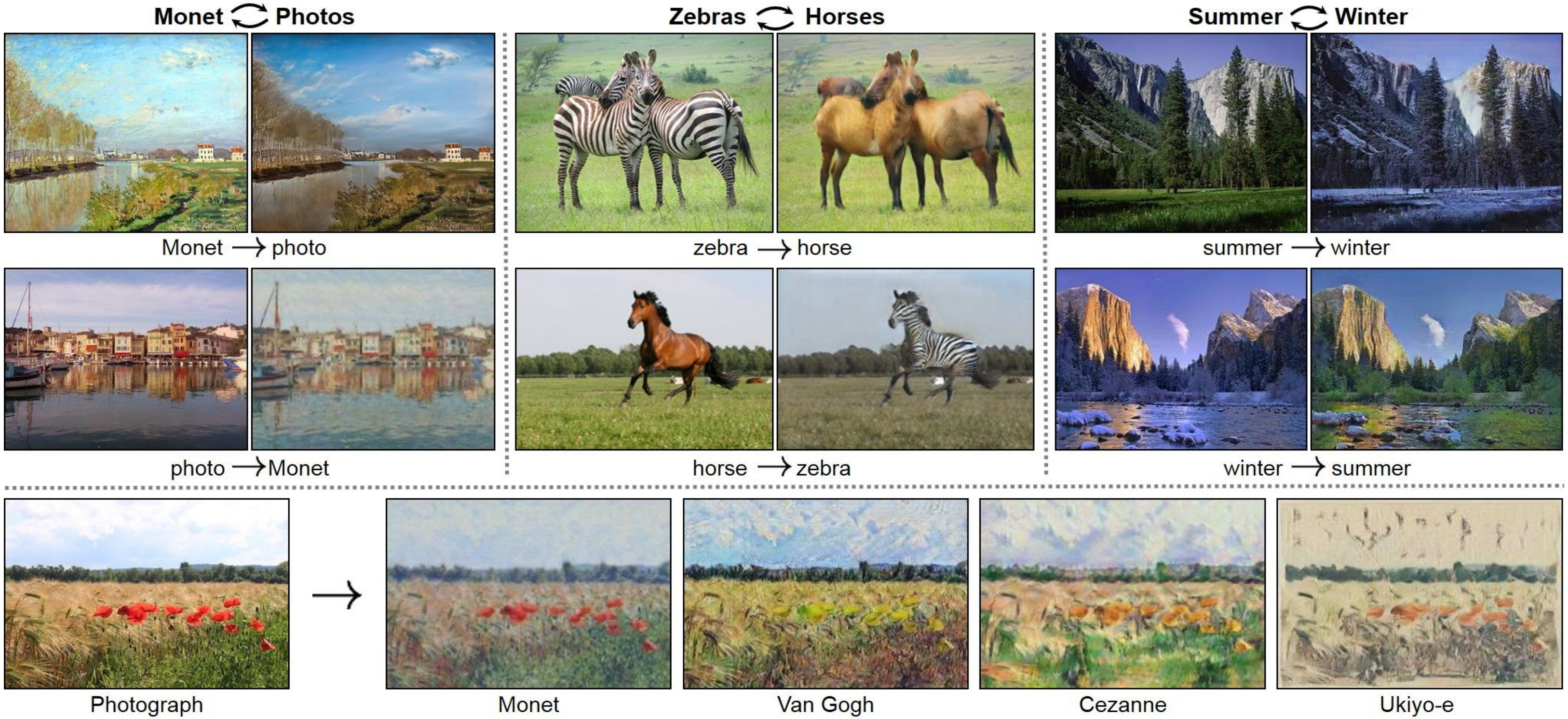


plain regression model is encouraged to “interpolate” outputs if uncertain

adversarial losses explicitly penalise outputs that look “too different from outputs in the training set”



Example Application 2:
Enabling **unsupervised** image-to-image training



“Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks” [Jun-Yan Zhu, Taesung Park, Phillip Isola, Alexei A. Efros]

Example Application 2:
Enabling **unsupervised** image-to-image training

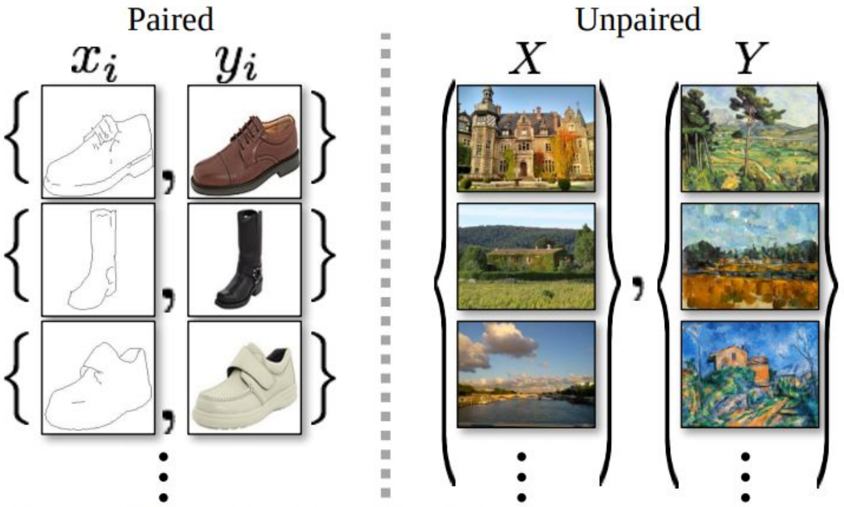
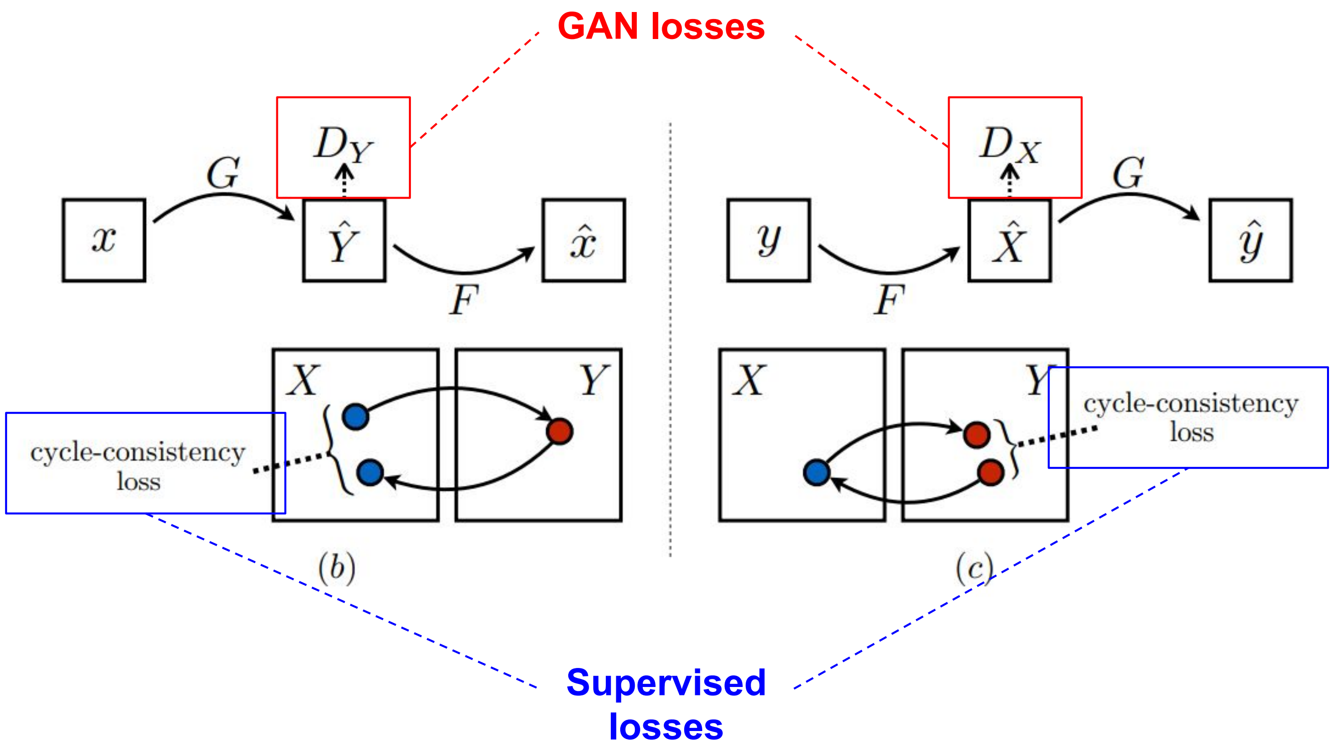


Figure 2: *Paired* training data (left) consists of training examples $\{x_i, y_i\}_{i=1}^N$, where the correspondence between x_i and y_i exists [22]. We instead consider *unpaired* training data (right), consisting of a source set $\{x_i\}_{i=1}^N$ ($x_i \in X$) and a target set $\{y_j\}_{j=1}$ ($y_j \in Y$), with no information provided as to which x_i matches which y_j.

“Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks” [Jun-Yan Zhu, Taesung Park, Phillip Isola, Alexei A. Efros]

Example Application 2:
Enabling **unsupervised** image-to-image training



"Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks" [Jun-Yan Zhu, Taesung Park, Phillip Isola, Alexei A. Efros]

Example Application 2:
Enabling **unsupervised** image-to-image training

“Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks” [Jun-Yan Zhu, Taesung Park, Phillip Isola, Alexei A. Efros]

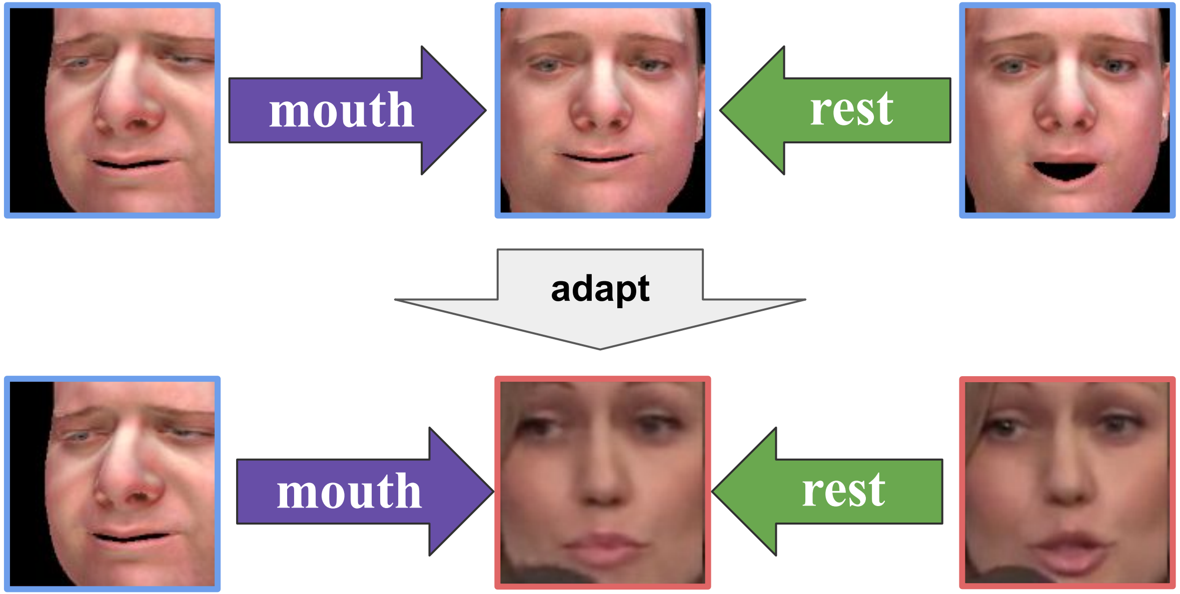
Example Application 2:
Enabling **unsupervised** image-to-image training

Takeaway:

adversarial losses enable discovery of latent correspondances in the structure of two datasets

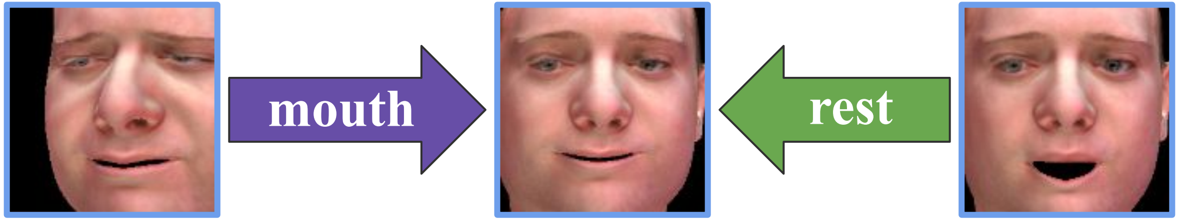
“Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks” [Jun-Yan Zhu, Taesung Park, Phillip Isola, Alexei A. Efros]

Example Application 3:
Cross-domain attribute manipulation



“PuppetGAN: Cross-Domain Image Manipulation by Demonstration” [B. Usman, N. Dufour, K. Saenko, C. Bregler]

Example Application 3: Cross-domain attribute manipulation

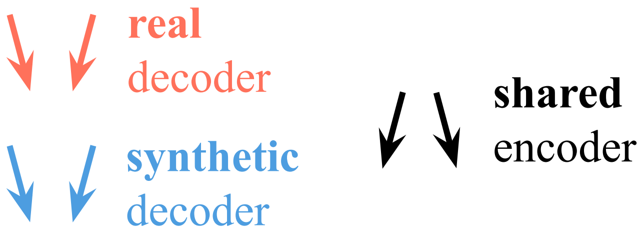


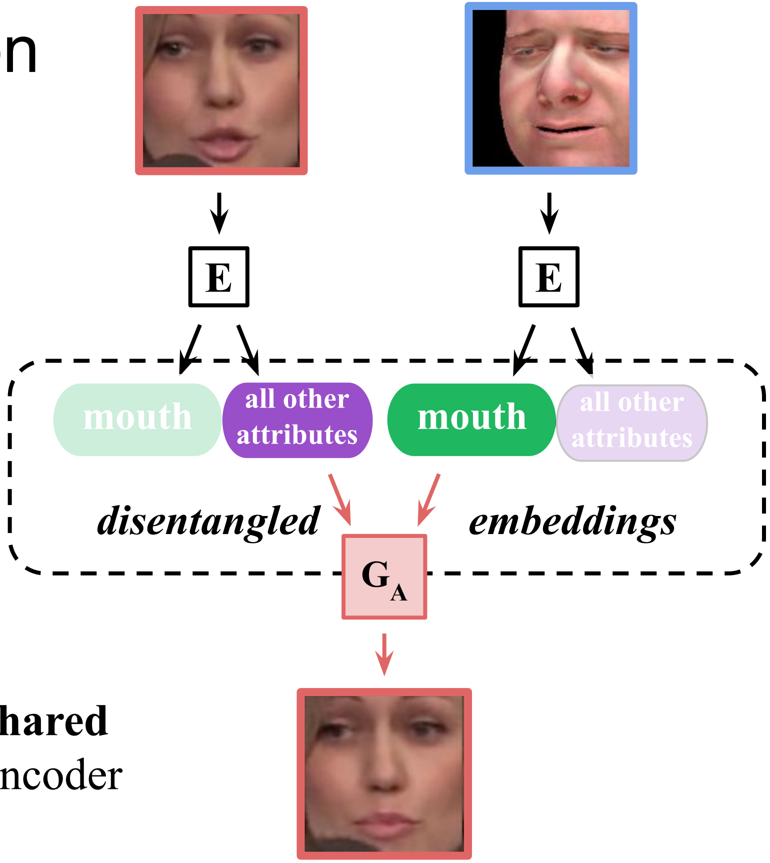
"PuppetGAN: Cross-Domain Image Manipulation by Demonstration" [B. Usman, N. Dufour, K. Saenko, C. Bregler]

Example Application 3:
Cross-domain attribute manipulation

We trained a **disentangled autoencoder**: we **split** the encoded vector into **two parts** and **force** one part to represent the attribute we manipulate (mouth) and other attributes (hair, mic, …).

How?

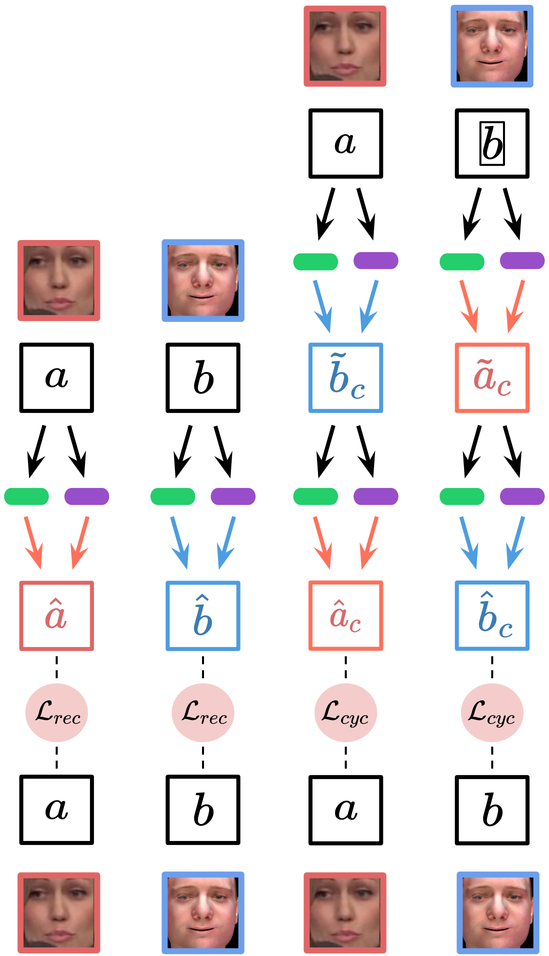




“PuppetGAN: Cross-Domain Image Manipulation by Demonstration” [B. Usman, N. Dufour, K. Saenko, C. Bregler]

Example Application 3:
Cross-domain attribute manipulation

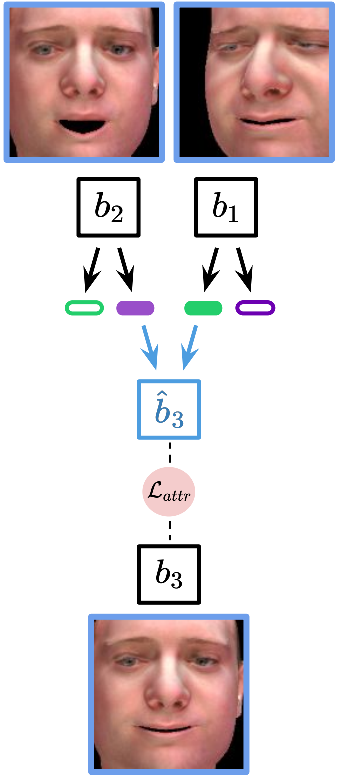
We combined **autoencoder** and **cycle losses** on both domains ...



“PuppetGAN: Cross-Domain Image Manipulation by Demonstration” [B. Usman, N. Dufour, K. Saenko, C. Bregler]

Example Application 3:
Cross-domain attribute manipulation

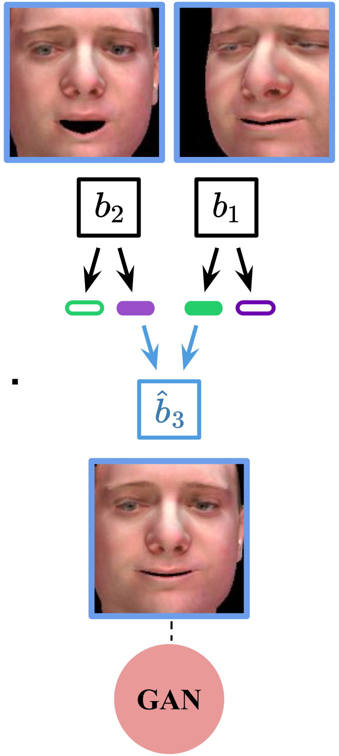
.. with **supervised losses** on synthetic data ...



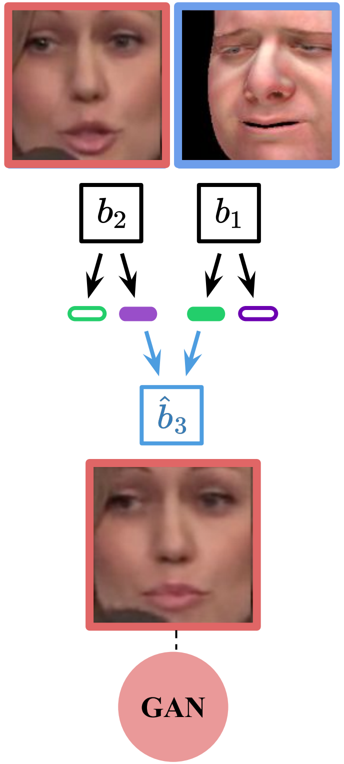
“PuppetGAN: Cross-Domain Image Manipulation by Demonstration” [B. Usman, N. Dufour, K. Saenko, C. Bregler]

Example Application 3:
Cross-domain attribute manipulation

... with **GAN losses** ...



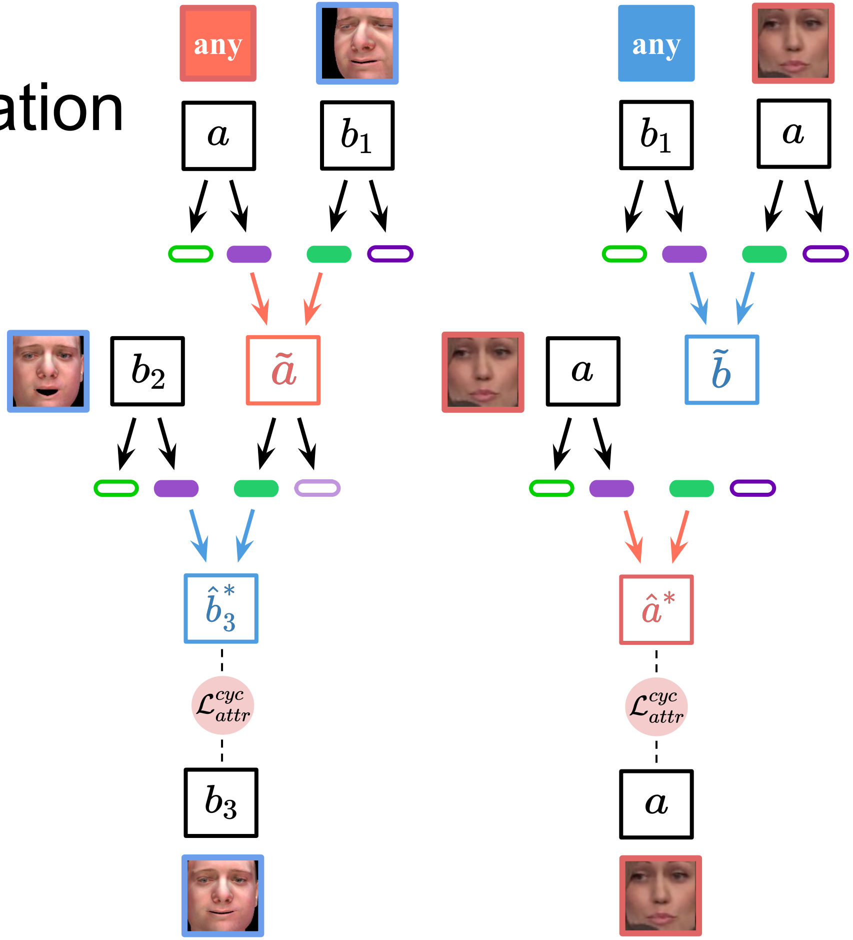
....



"PuppetGAN: Cross-Domain Image Manipulation by Demonstration" [B. Usman, N. Dufour, K. Saenko, C. Bregler]

Example Application 3:
Cross-domain attribute manipulation

... and **compositional constraint losses** to ensure that all components are used.



"PuppetGAN: Cross-Domain Image Manipulation by Demonstration" [B. Usman, N. Dufour, K. Saenko, C. Bregler]

Example Application 3:
Cross-domain attribute manipulation

Takeaway:

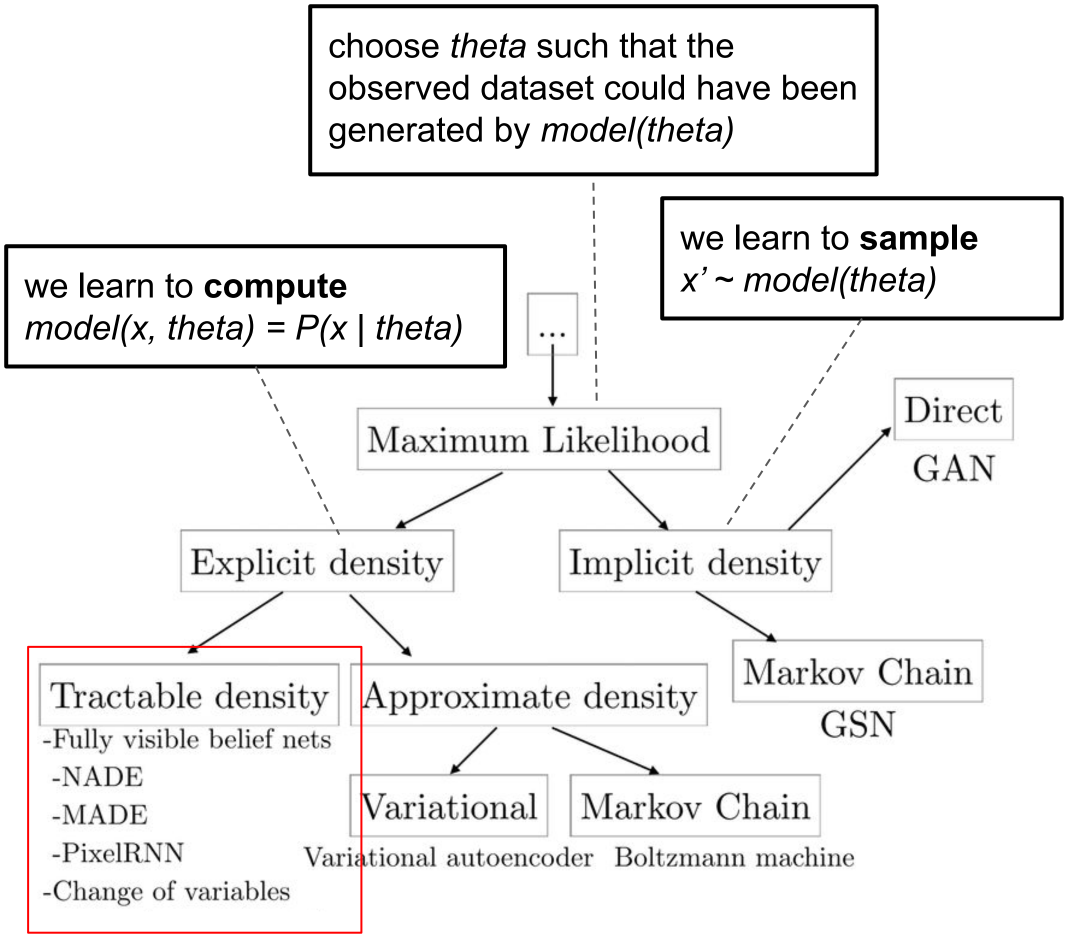
adversarial losses enable “forcing” the model to store
information necessary for reconstructing
specific “aspects” of the input image
at specific dimensions of the latent code

“PuppetGAN: Cross-Domain Image Manipulation by Demonstration” [B. Usman, N. Dufour, K. Saenko, C. Bregler]

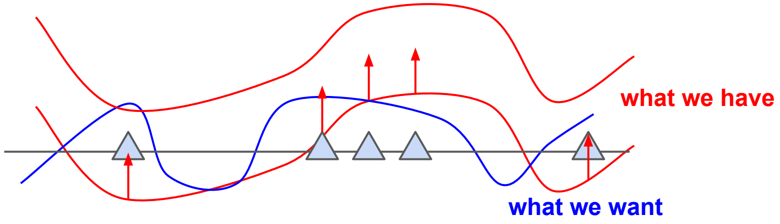
Outline

1. Some applications of convolutional autoencoders and GAN
    a. Image-to-Image Translation with Conditional Adversarial Nets
    b. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks
    c. PuppetGAN: Cross-Domain Image Manipulation by Demonstration

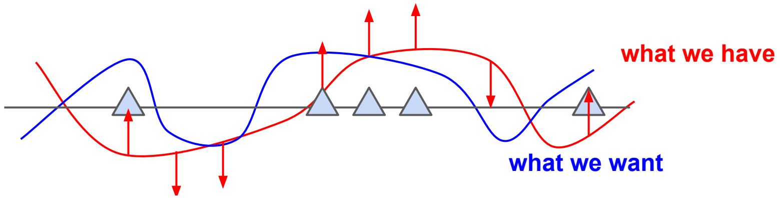
2. **Normalizing Flows**
    a. Change of variable formula
    b. Planar and radial flows
    c. Real NVP
    d. GLOW
    e. FFJORD
    f. Likelihood vs probability


choose *theta* such that the observed dataset could have been generated by *model(theta)*
we learn to **compute**
model(x, theta) = P(x | theta)
we learn to **sample**
x' ~ model(theta)
...
Maximum Likelihood
Explicit density
Implicit density
Direct
GAN
Tractable density
-Fully visible belief nets
-NADE
-MADE
-PixelRNN
-Change of variables
Approximate density
Markov Chain
GSN
Variational
Variational autoencoder
Markov Chain
Boltzmann machine

How to train a data model from positive samples only?



If we trained a neural network f(x; θ) to have high values at our training points x_i, it could just shift everything upwards.



We could train a GAN to generate “negative” samples, but the whole procedure becomes fragile.

We could use a model with a “fixed budget”, i.e. an autoencoder (# points it can “remember”) or a density models (integrates to one).

Why train a model from positive samples only?

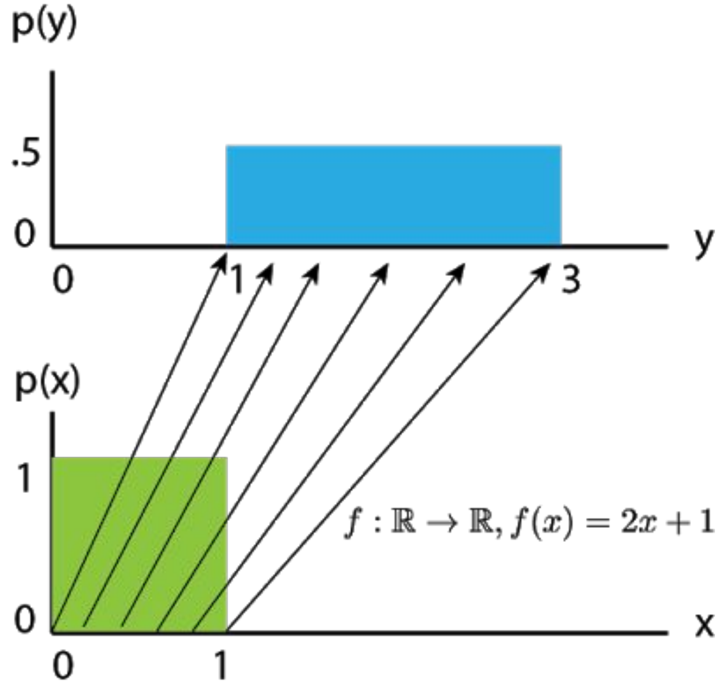
1. **Adversarial Robustness:** If the input X is not from the training distribution P(X), refuse classification
2. **Detecting Data Shift:** if P(X) shifted over time, retrain the model
3. **Outlier Detection:** detect abnormalities in observed data
4. **“Learned” data priors:** improved image synthesis or structure in segmentation maps

Background: change of variable formula

U[1, 3] **P(T(0.5)) =**
= P(2) = 0.5

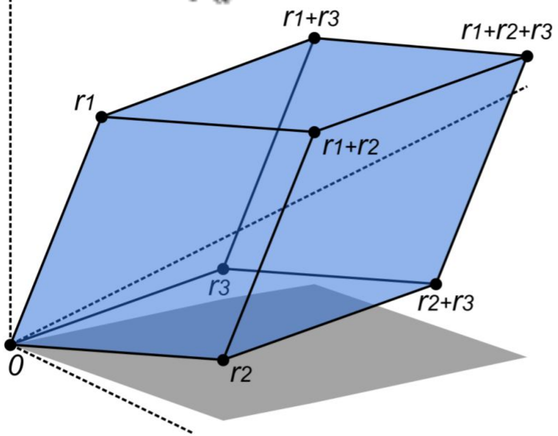
T(x) = 2x + 1

U[0, 1] **P(0.5) = 1**

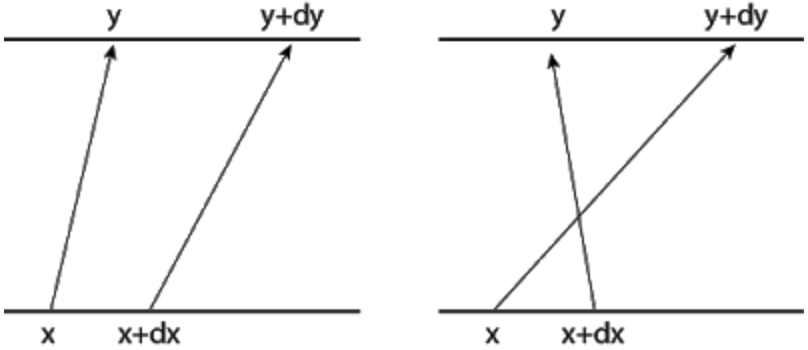


Background: change of variable formula

$$\int_{\varphi(a)}^{\varphi(b)} f(u)\,du = \int_a^b f(\varphi(x))\varphi'(x)\,dx.$$



$$\frac{dy}{dx} > 0 \qquad \frac{dy}{dx} < 0$$



p(x) dx = p(y) dy

p(y) = p(x) **det | (dT^{-1}(y)/dy) |**

log p(y) = log p(x) + log det | (dT^{-1}(y)/dy) |

Background: change of variable formula

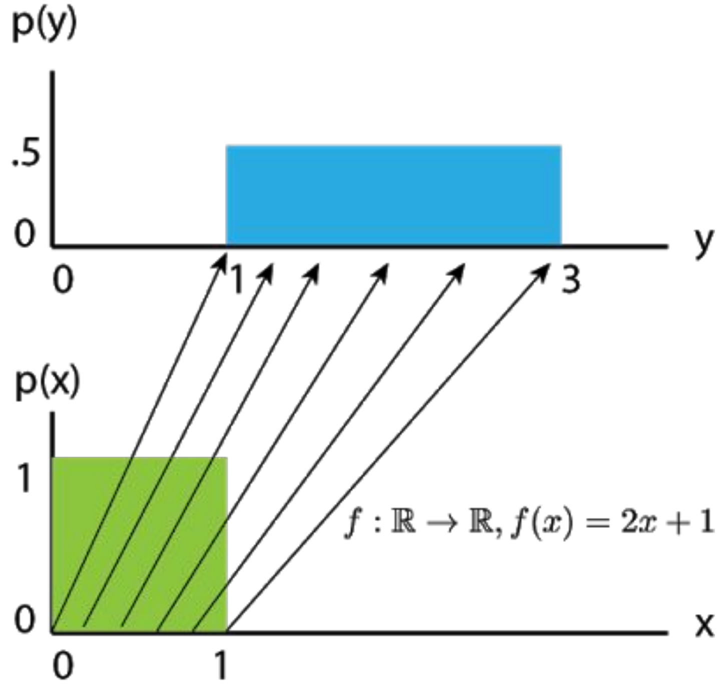
U[1, 3] **P(T(0.5)) =**
= P(2) = 0.5

T(x) = 2x + 1
$T^{-1}(y) = y/2 - ½$
$dT^{-1}(y)/dy = ½$

U[0, 1] **P(0.5) = 1**

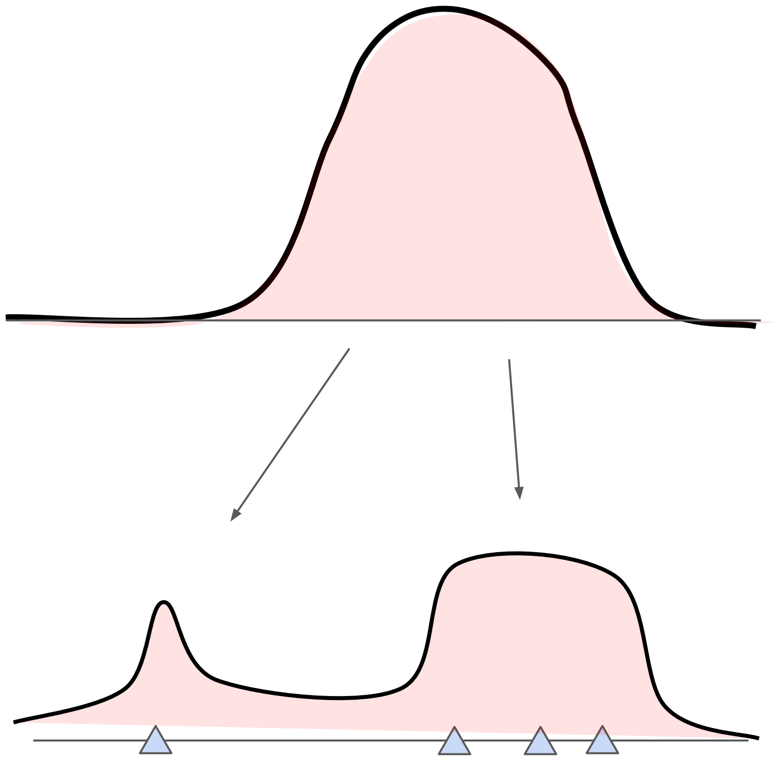
$p_Y(y) = p_X(T^{-1}(y))$ * **$\det | (dT^{-1}(y)/dy) |$**

$p_Y(y) = I[0 < (y/2 - ½) < 1] * ½$
$= I[1 < y < 3] * ½$

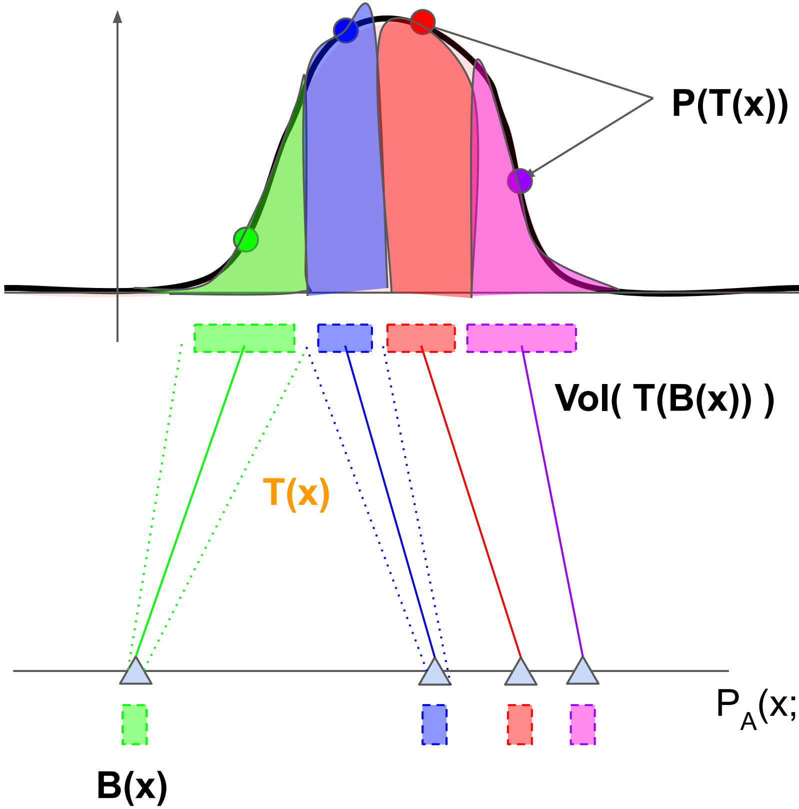


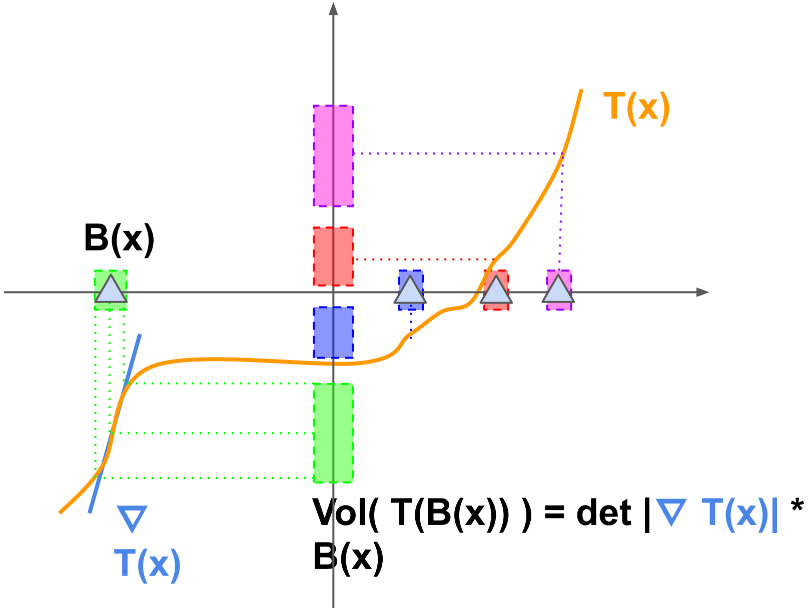
Normalizing flows for density estimation

like shuffling a sand castle - we move sand around to increase the amount of send near data points, but the total amount of the sand stays constant



Normalizing flows for density estimation

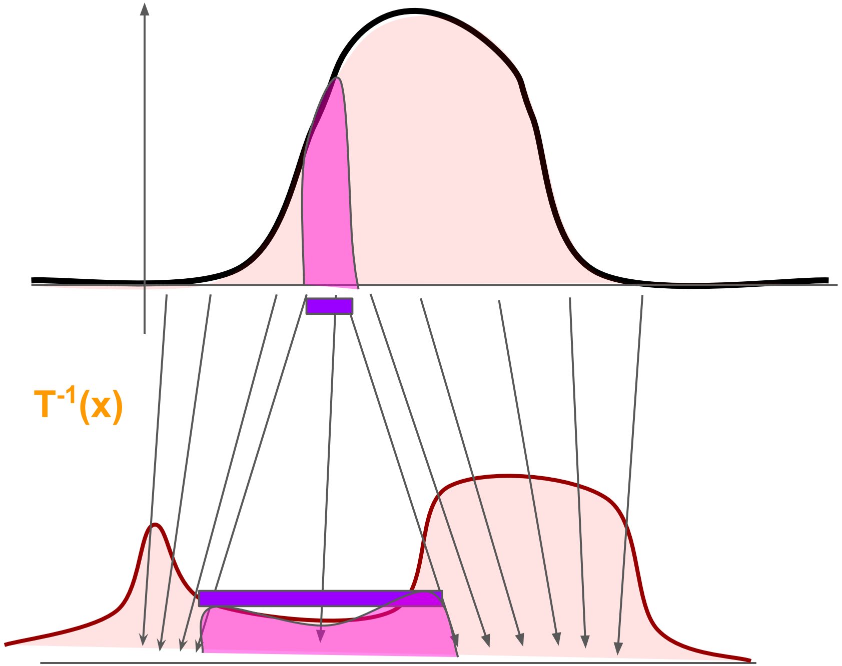


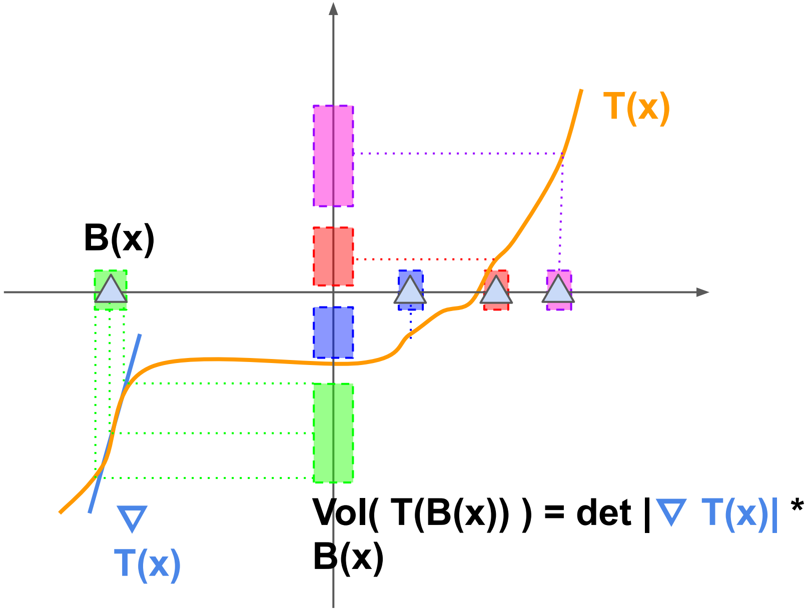


$P_A(x; T) = P(T(x)) * \det | \nabla T(x) | * \mathrm{Vol}(B(x))$

let’s solve the problem “backwards”

Normalizing flows for density estimation





$P_A(x; F) = P(T(x)) * \det | \nabla T(x) | * Vol(T(x))$

Low probability areas are “stretches”

In order to define a normalizing flow model we need

1. A **one-to-one** mapping $F(x,\boldsymbol{\theta})$: $\mathbf{R}^n \rightarrow \mathbf{R}^n$
2. $F^{-1}(x,\boldsymbol{\theta})$
3. $\det[\text{Jac } F(x,\boldsymbol{\theta})]$

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial x_1} & \cdots & \dfrac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix}.$$

Planar Flow

$$f(\mathbf{z}) = \mathbf{z} + \mathbf{u}h(\mathbf{w}^T\mathbf{z} + b), \tag{4}$$

with $\mathbf{u}, \mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$ and h an element-wise non-linearity. Let $\psi(\mathbf{z}) = h'(\mathbf{w}^T\mathbf{z} + b)\mathbf{w}$. The determinant can be easily computed as

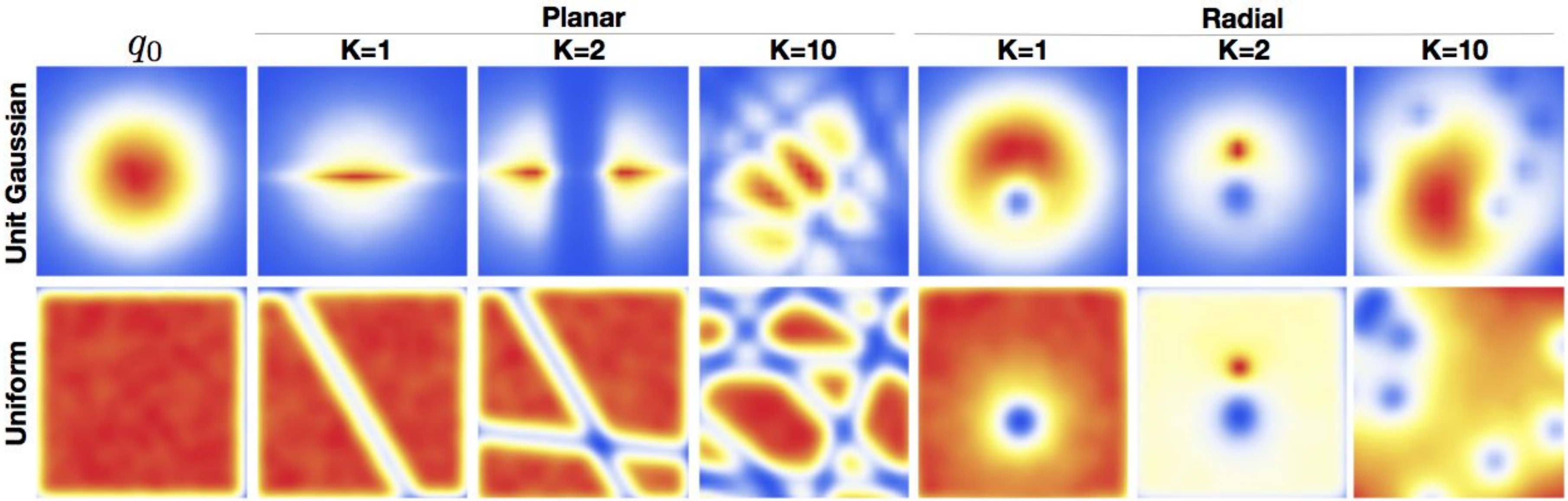
$$\left|\det \frac{\partial f}{\partial \mathbf{z}}\right| = \left|1 + \mathbf{u}^T\psi(\mathbf{z})\right|. \tag{5}$$

We can think of it as slicing the $\mathbf{z}$-space with straight lines (or hyperplanes), where each line contracts or expands the space around it, see figure 1.

Radial Flow

$$f(\mathbf{z}) = \mathbf{z} + \beta h(\alpha, r)(\mathbf{z} - \mathbf{z}_0), \tag{6}$$

with $r = \|\mathbf{z} - \mathbf{z}_0\|_2$, $h(\alpha, r) = \frac{1}{\alpha + r}$ and parameters $\mathbf{z}_0 \in \mathbb{R}^d, \alpha \in \mathbb{R}_+$ and $\beta \in \mathbb{R}$.



$$\mathbf{z}_K = f_K \circ \cdots \circ f_1(\mathbf{z}_0), \quad \mathbf{z}_0 \sim q_0(\mathbf{z}_0),$$

$$\mathbf{z}_K \sim q_K(\mathbf{z}_K) = q_0(\mathbf{z}_0) \prod_{k=1}^{K} \left| \det \frac{\partial f_k}{\partial \mathbf{z}_{k-1}} \right|^{-1}.$$

to learn more complex distributions, apply multiple flows in a row

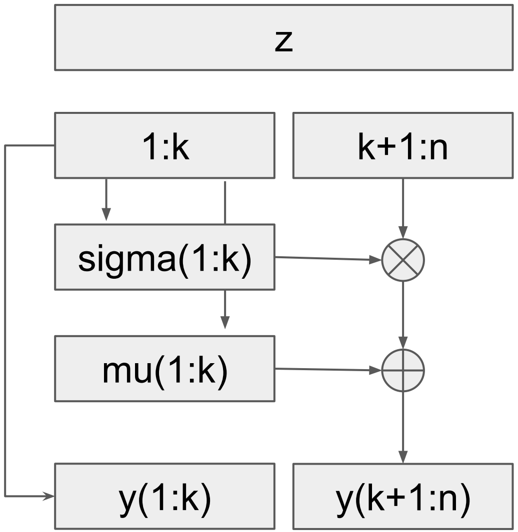
Real Non-Volume Preserving Flows (R-NVP)

$$\mathbf{y}_{1:k} = \mathbf{z}_{1:k},$$
$$\mathbf{y}_{k+1:d} = \mathbf{z}_{k+1:d} \circ \sigma(\mathbf{z}_{1:k}) + \mu(\mathbf{z}_{1:k}).$$

$$\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

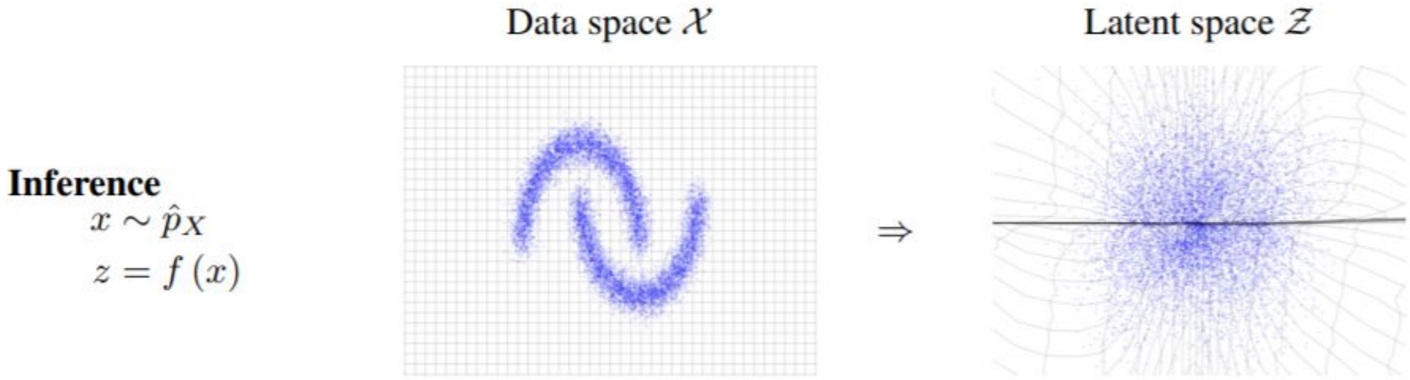
$$\det \frac{\partial \mathbf{y}}{\partial \mathbf{z}} = \prod_{i=1}^{d-k} \sigma_i(\mathbf{z}_{1:k}).$$

“affine coupling”

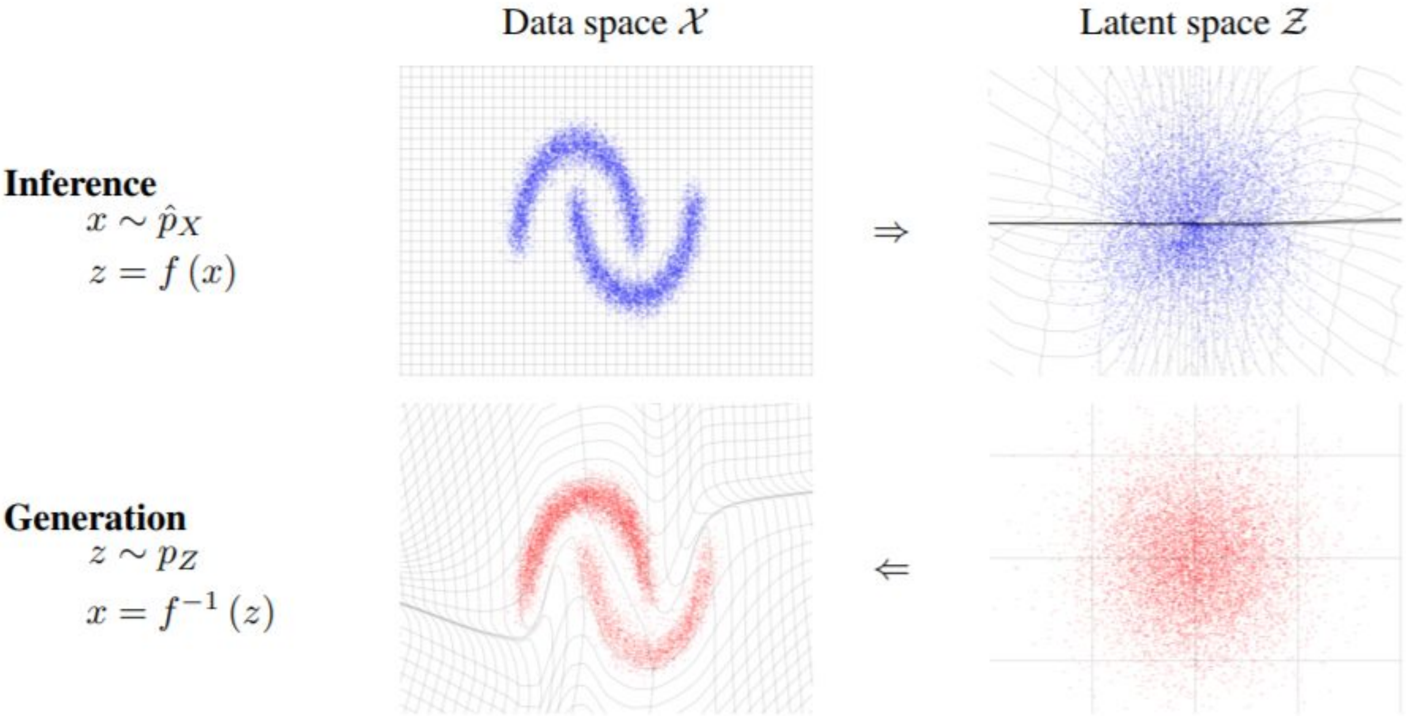


“Density estimation using Real NVP” by Laurent Dinh, Jascha Sohl-Dickstein, Samy Bengio

Real Non-Volume Preserving Flows (R-NVP)

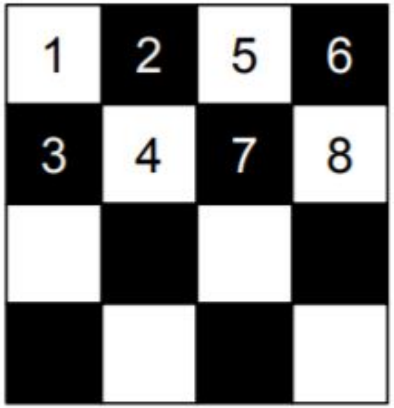


Real Non-Volume Preserving Flows (R-NVP)

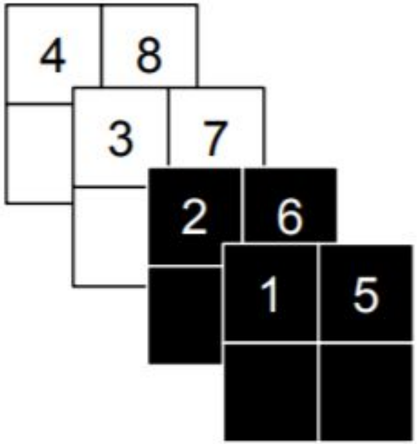


Real Non-Volume Preserving Flows (R-NVP)

checkerboard swap

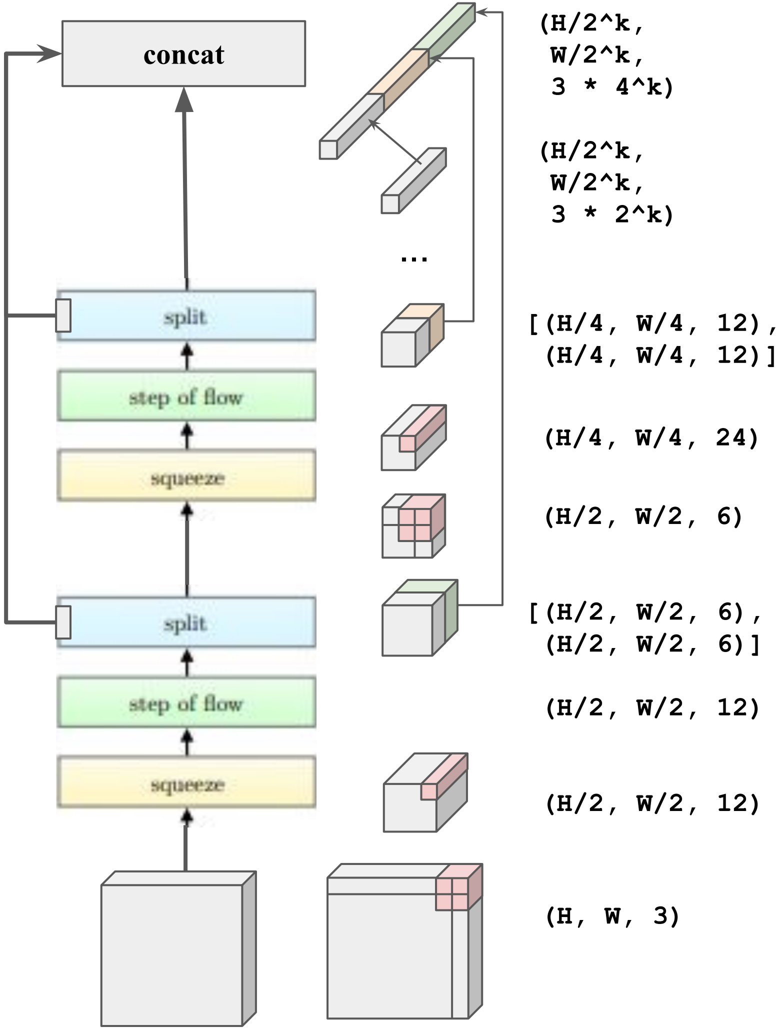


“squeeze”



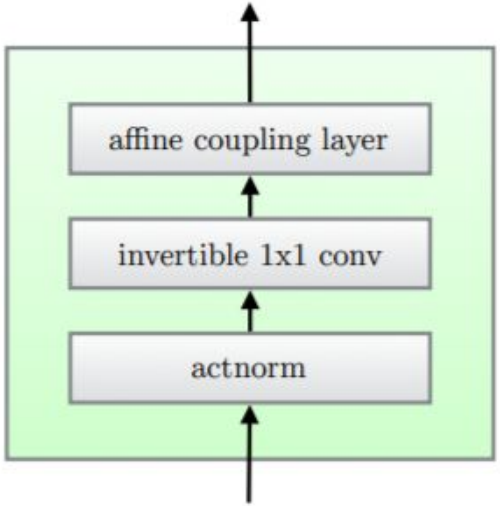
Real Non-Volume Preserving Flows (R-NVP)

Glow: Generative Flow with Invertible 1x1 Convolutions

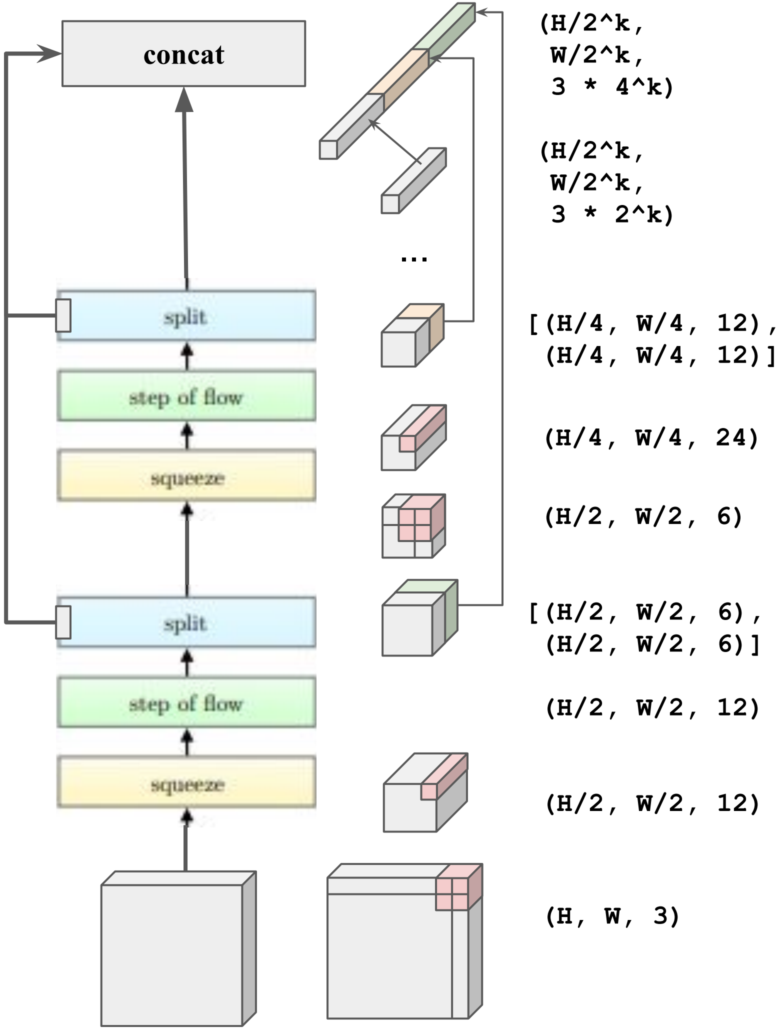


= very deep RealNVP
+ invertible 1x1 conv instead of swap
+ multiscale features

Glow: Generative Flow with Invertible 1x1 Convolutions



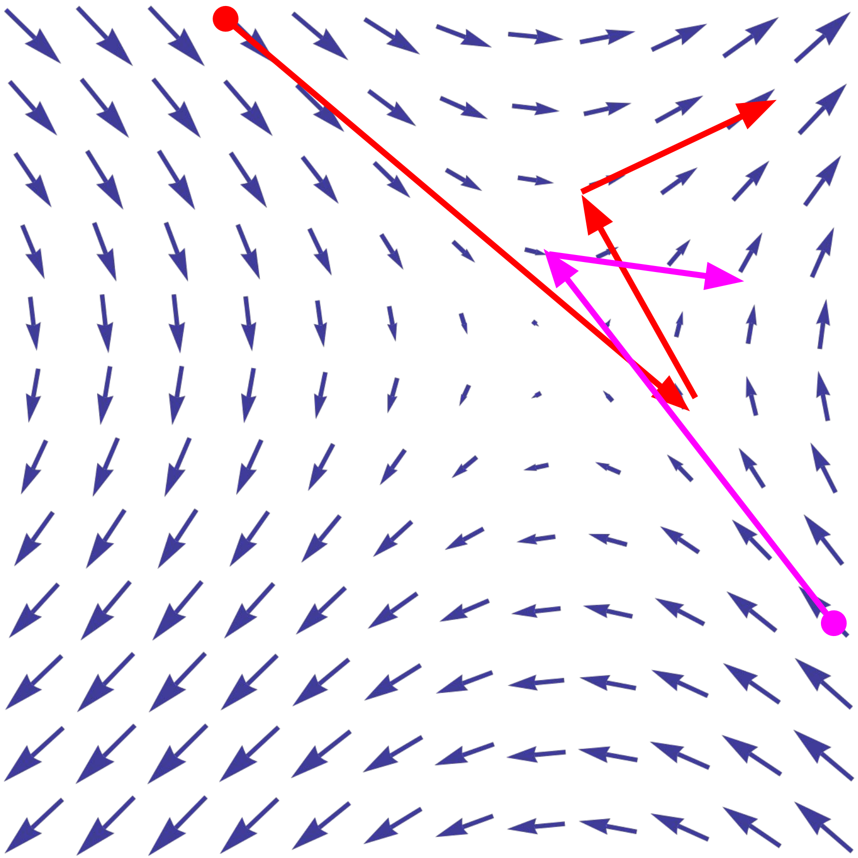
= very deep R-NVP
+ invertible 1x1 conv instead of swap
+ multiscale features



Neural ODE and FFJORD

$x(t_0) = x_0$
$x'(t) = \mathbf{f}(x(t))$
$x(t_1) = ?$

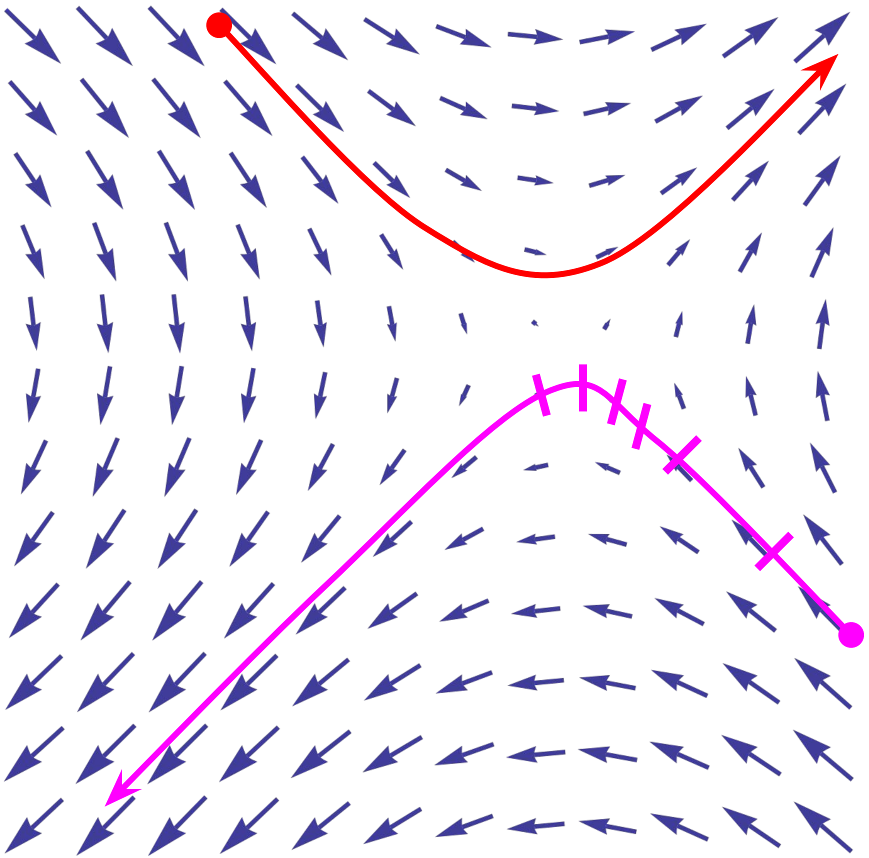
What if we use an **explicit method** with **fixed** step size?



Neural ODE and FFJORD

$x(t_0) = x_0$
$x'(t) = \mathbf{f}(x(t))$
$x(t_1) = ?$

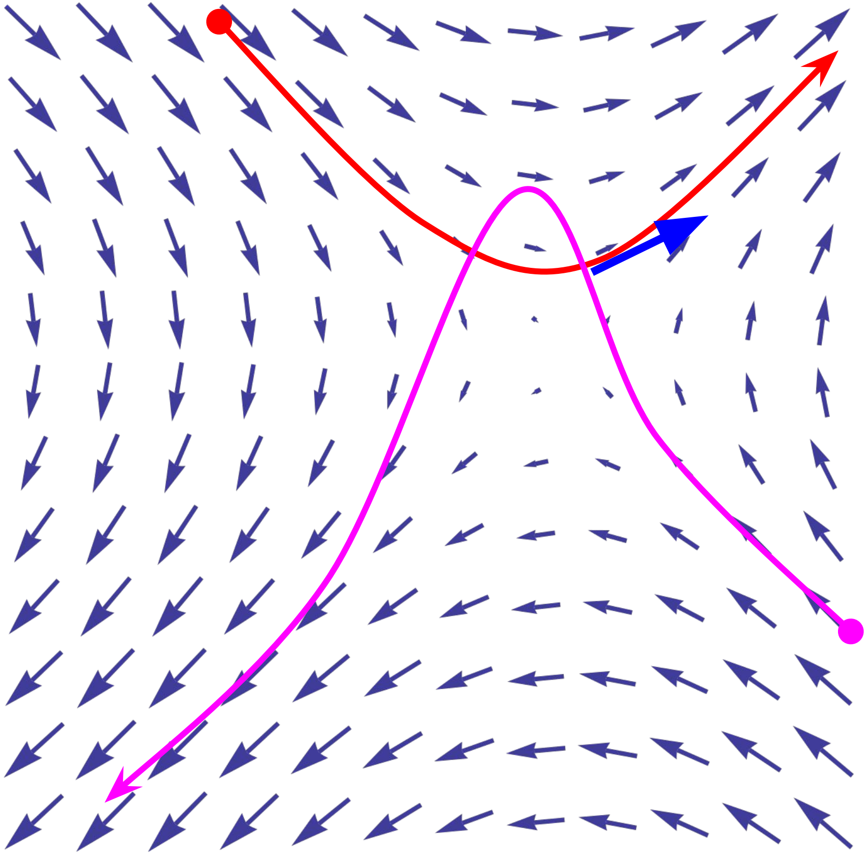
If we use a "proper solver", we get adaptive step size.



Neural ODE and FFJORD

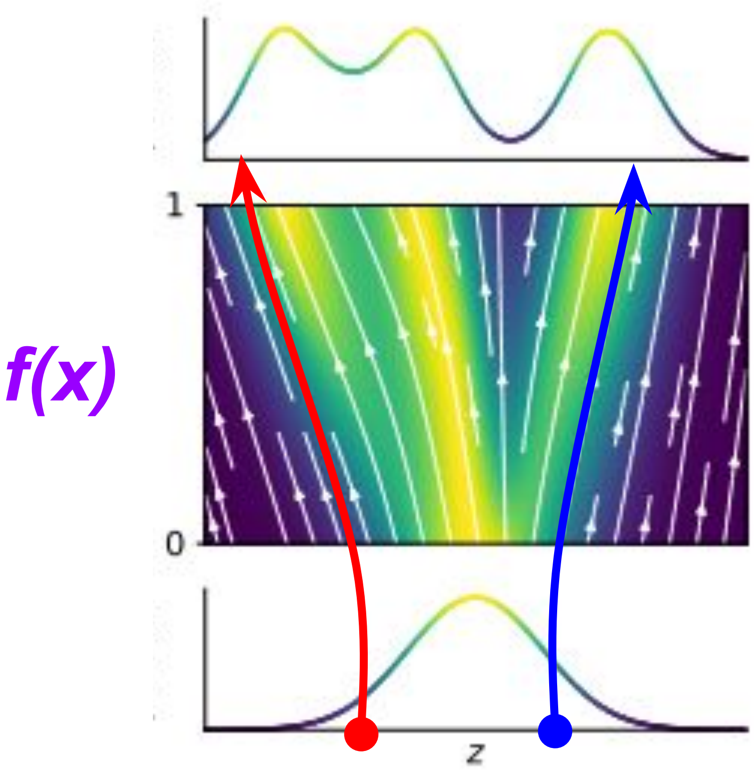
$x(t_0) = x_0$
$x'(t) = \mathbf{f}(x(t))$
$x(t_1) = ?$

If we use a "proper solver", we get adaptive step size.



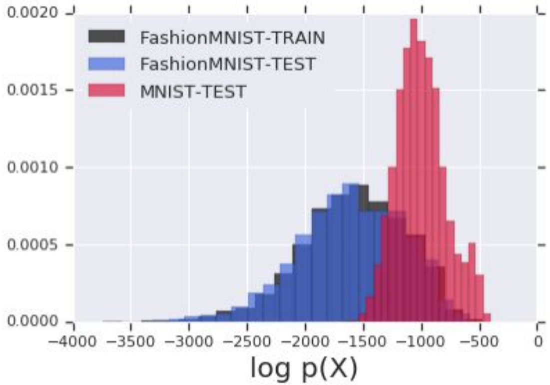
impossible!

Neural ODE and FFJORD

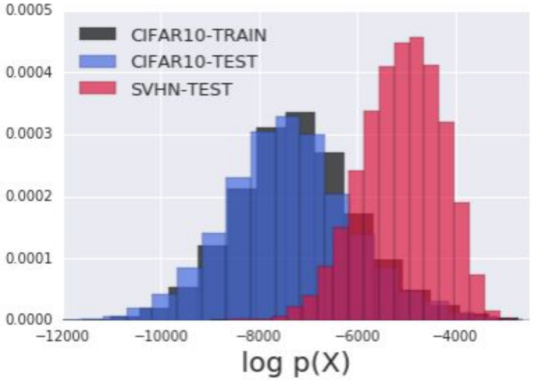


FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models

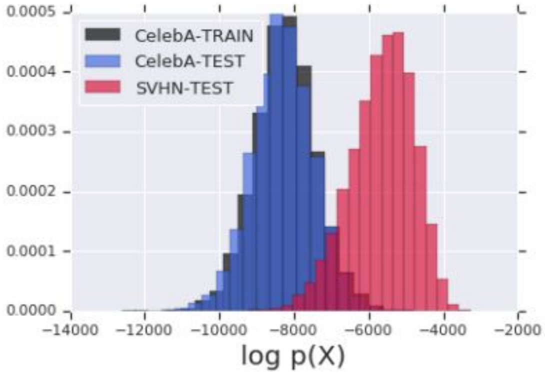
“Do Deep Generative Models Know What They Don't Know?”, ICLR’19



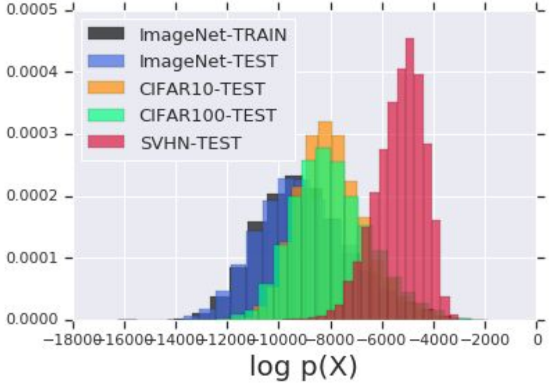
a) Train on FashionMNIST, Test on MNIST



(b) Train on CIFAR-10, Test on SVHN



(c) Train on CelebA, Test on SVHN



(d) Train on ImageNet,
Test on CIFAR-10 / CIFAR-100 / SVHN

If model $P_A(x)$ is **trained** on A then for many datasets B

$P_A(\mathbf{B}) > P_A(\mathbf{A})$

which is quite counter-intuitive …

and we never see anything like B if we sample from P(A) ..

High likelihood of X does not mean that X is likely!

$$A = \{y \in \mathbb{R}^{100} \mid ||y|| \leq \varepsilon\}$$

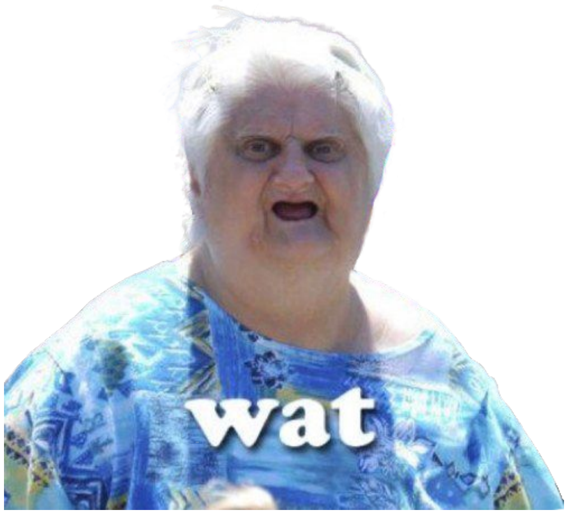
$$P(A) = \int_A \mathcal{N}_{100}(x; 0, I)dx$$

- the **probability** of point being in set A is **low**

$$B = \{x_i \in \mathbb{R}^{100} \mid ||x|| \leq \varepsilon\}_{i=0}^{N}$$

$$\mathcal{L}(B) = \frac{1}{N}\sum_{x_i \in B} \log \mathcal{N}_{100}(x; 0, I)$$

- but the mean **likelihood** of points from this set is **high**



Final takeaways

1. GANs can help to learn and use the structure of the output domain
2. Normalizing Flows enable density estimation in higher dimensions