UNIVERSITY OF TORONTO

Unsupervised Learning

With Autoencoders and Likelihood-based Generative Models

Jörn-Henrik Jacobsen

Disclaimer: Not exhaustive, no GANs, no self-supervised learning

Vector Institute and University of Toronto

Recap: Supervised Learning

Deep Learning's biggest success story!

Supervised learning: Learn F_θ that maps inputs to ***known*** targets

Dataset of input/target pairs: $$D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$$

Training objective: $$L(\theta, D) = \sum_{(x,y)\sim D} -\log p(y \,|\, x, \theta)$$

Maximum Likelihood is dominant paradigm for supervised learning

Recap: Supervised Learning

Deep Learning's biggest success story!

Supervised learning: Learn F_θ that maps inputs to ***known*** targets

F_θ

Flower

Elephant

Ship

Pumpkin

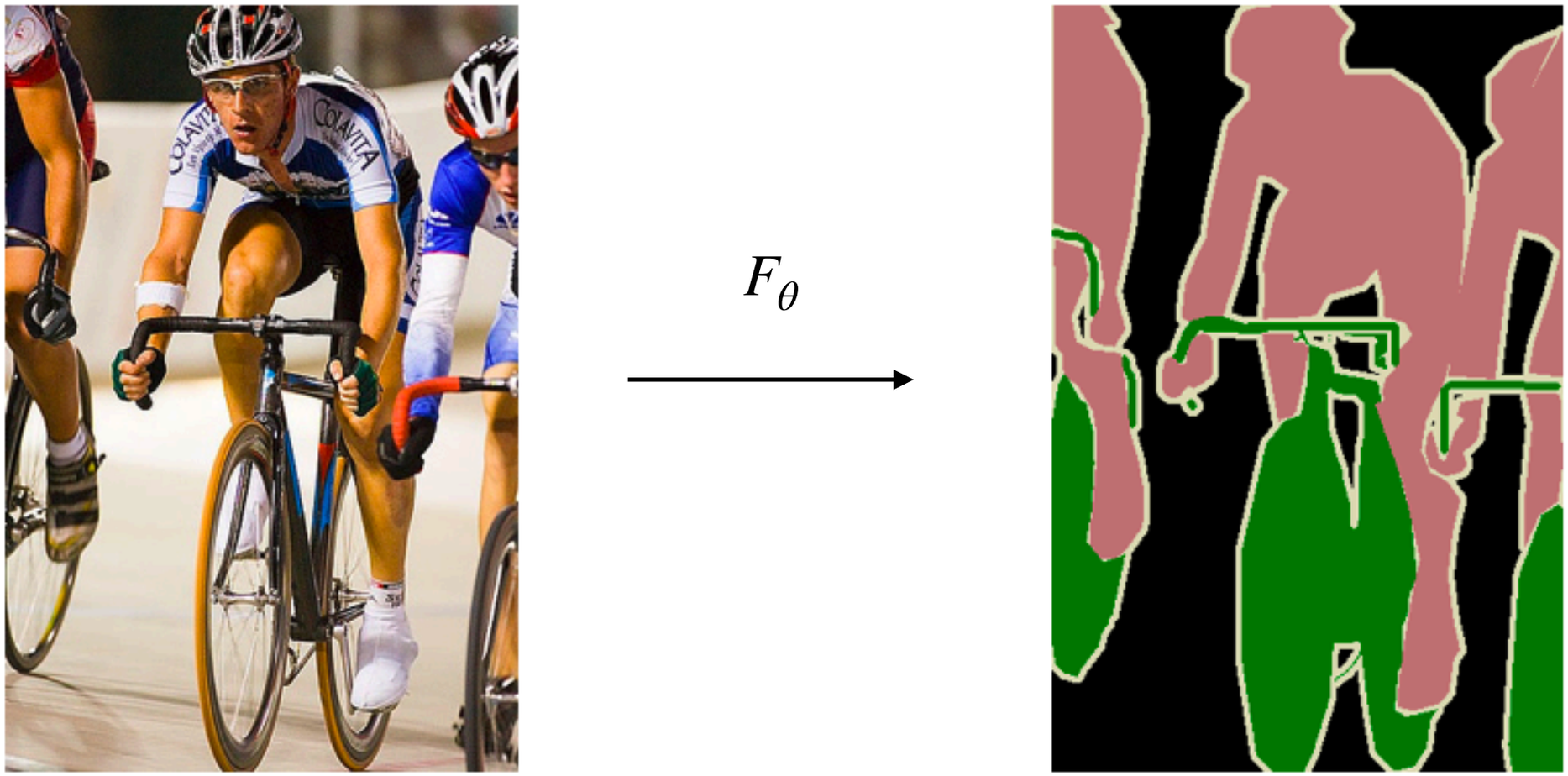
Dog

Image Classification

Recap: Supervised Learning

Deep Learning's biggest success story!

Supervised learning: Learn F_θ that maps inputs to ***known*** targets



Semantic Segmentation

Recap: Supervised Learning

Deep Learning's biggest success story!

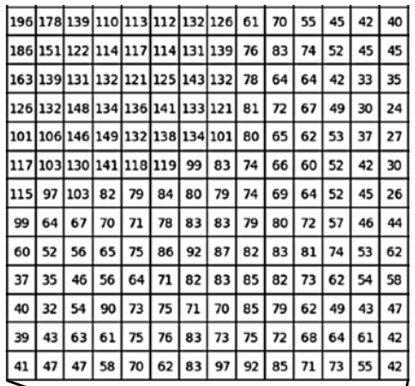
Supervised learning: Learn F_θ that maps inputs to ***known*** targets

$$F_\theta \longrightarrow$$

"construction worker in orange safety vest is working on road."

Image Captioning … (and many more)

Recap: Supervised Learning

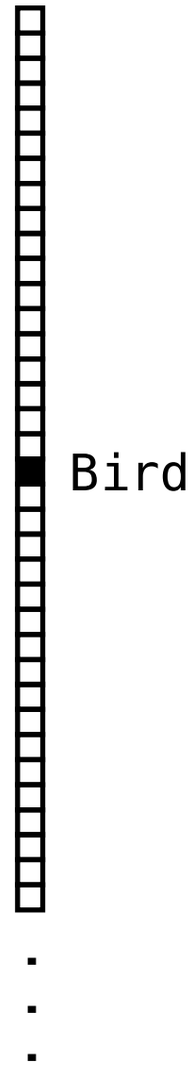


Inputs

$X \in \mathbb{R}^D$

$D \approx 150.000$

F_θ



Targets

$Y \in \mathbb{R}^d$

$d = 1.000$

Inputs are high-dimensional, targets low-dimensional

Lots of variability needs to be discarded!

Maximum likelihood training allows to learn what to discard from input target pairs

Issues with Supervised Learning

Supervised learning only requires narrow understanding of input distribution, *just enough to extract labels reliably from train set*

Narrow understanding can lead to poor generalization if datasets and tasks are not large and diverse enough

Labels can be expensive, many examples where it is difficult to obtain high-quality input-target pairs (e.g. labeling pathologies in medical images)

Promise of Unsupervised Learning

Make use of fact that unlabeled data are abundant!

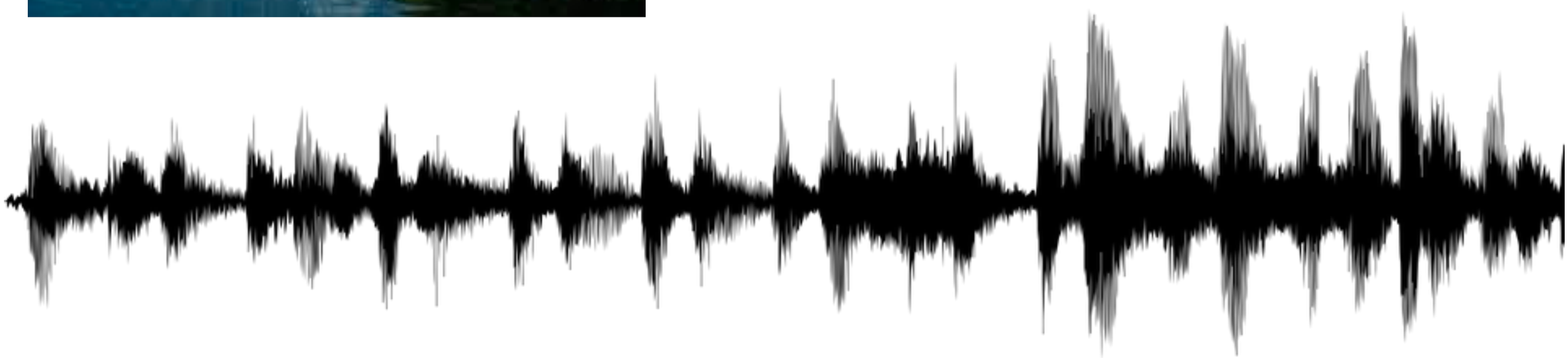
Core idea of unsupervised learning in a nutshell:

- Learn as much knowledge from unlabeled data as possible
- Resulting model has gained holistic understanding of training data from where adaptation to new tasks is easy

Sounds great but how do we extract this knowledge from unlabeled data?

Issues with **UN**supervised Learning

4 For testing the alignment: Here is a colored sample sentence.
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

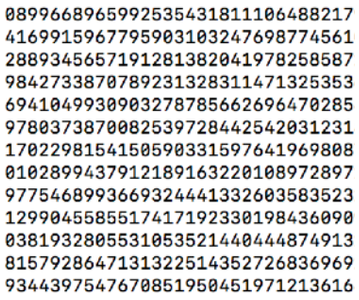


Inputs are high-dimensional with lots of structure and unclear what variability to pay attention to!

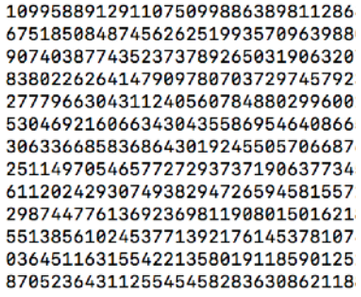
Can’t just discard variability, but need general objective which formalizes “summarizing” the data.

UNsupervised Learning

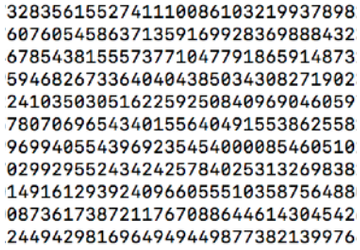
x_1



x_2



x_3



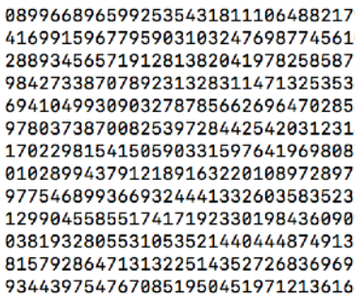
.
.
.

Starting with dataset *without* targets:

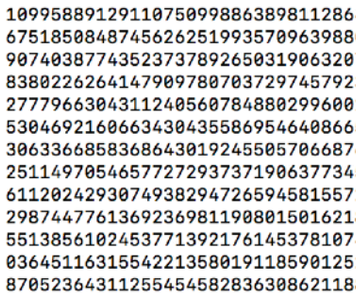
$$D = \{x_1, \ldots, x_N\}$$

UNsupervised Learning

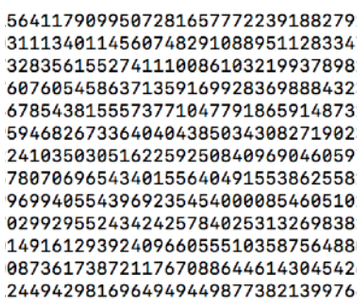
x_1



x_2



x_3



.
.
.

Starting with dataset *without* targets:

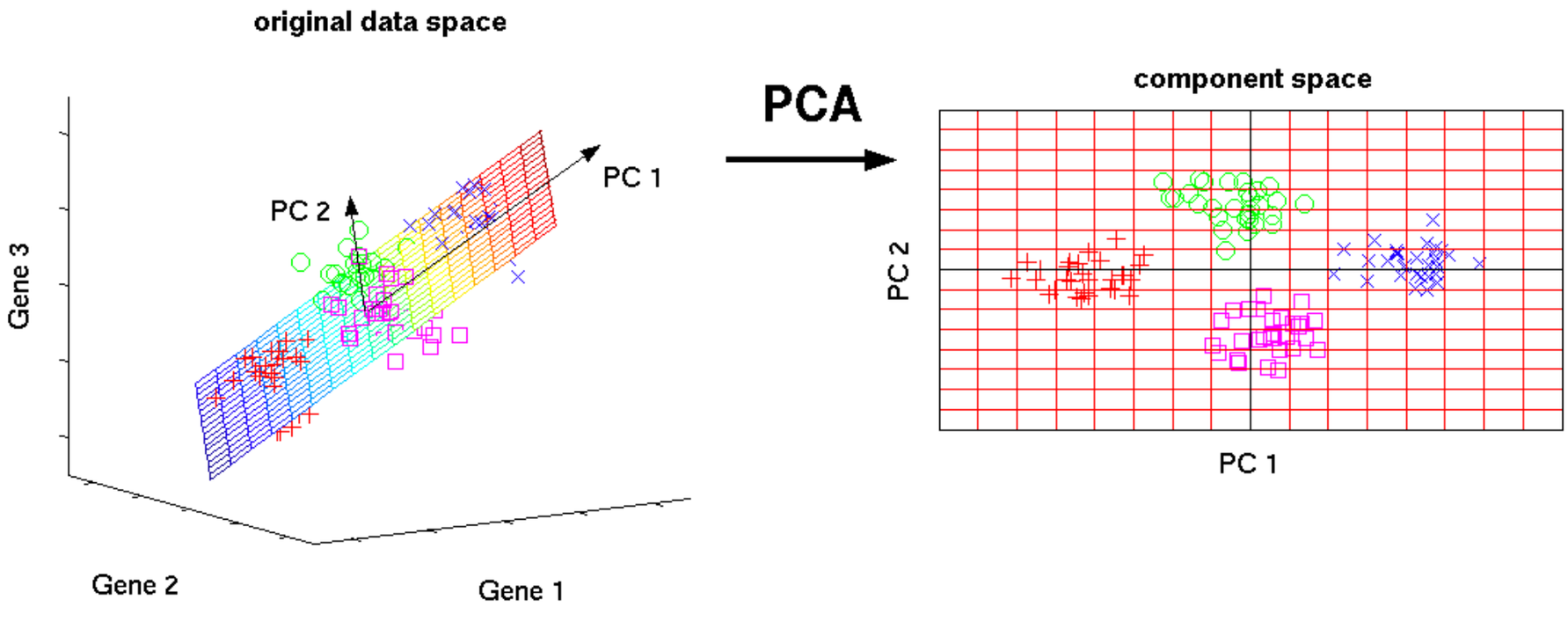
$$D = \{x_1, \ldots, x_N\}$$

One option, learn compressed representation of data.

Project data to lower-dimensional manifold and represent it by location on this manifold.

Core idea of Principal Component Analysis.

Principal Component Analysis

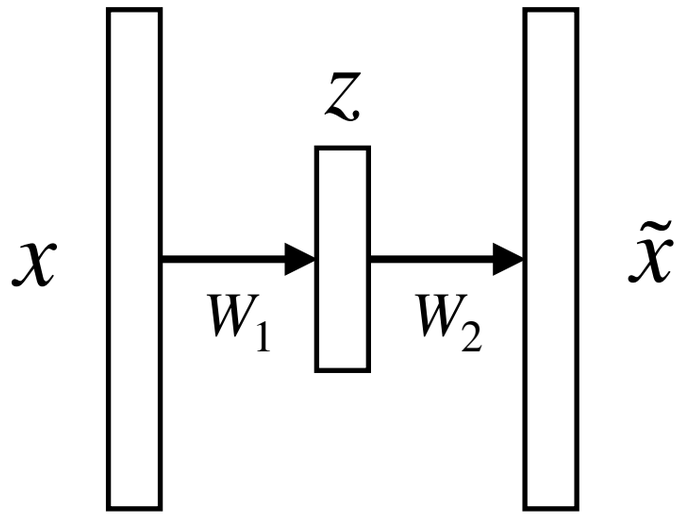


PCA represents D-dimensional data in linear N-dimensional subspace

Project datapoints onto subspace spanned by orthogonal components in directions where data has highest variance

Loses information about location in remaining orthogonal directions, not a problem because variance was low in them -> *Little information*

Linear Autoencoder



$$L(\theta, \phi, D) = \sum_{x \sim P_{data}} [x - W_2 W_1 x]^2$$

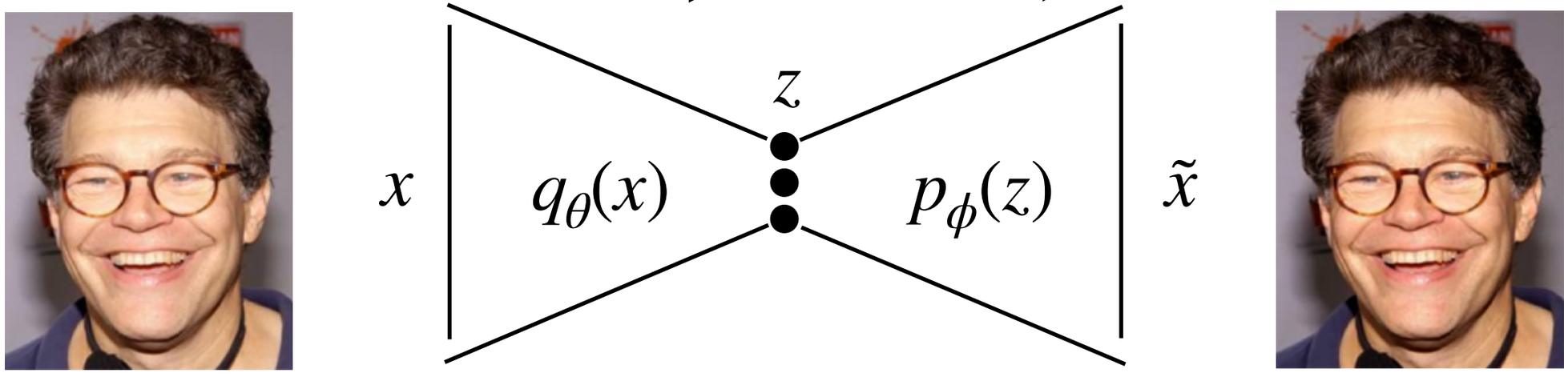
Train with mean squared reconstruction error via backpropagation.

If bottleneck is lower dimensional than data, result is equivalent to PCA!

However, bottleneck units will most likely not correspond to principal components. They span same space, but may be rotated and skewed.

Advantage: Can be generalized to non-linear case!

Non-linear Autoencoders

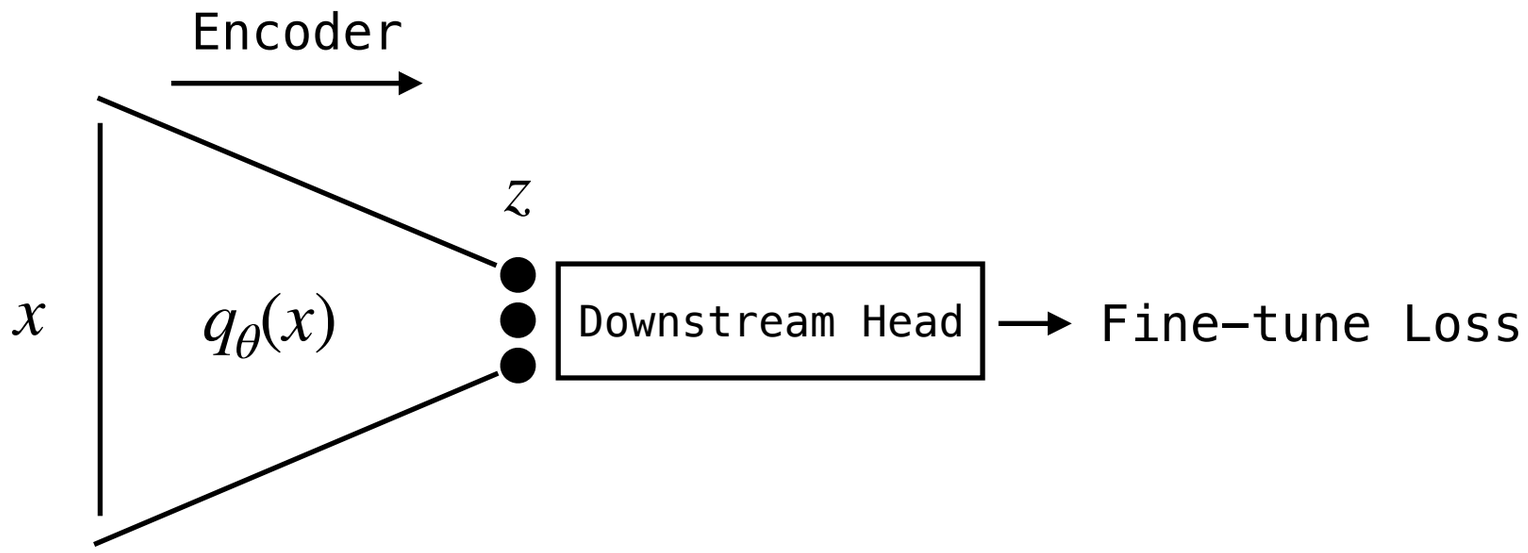


$$L(\theta, \phi, D) = \sum_{x \sim P_{data}} [x - p_\theta(q_\phi(x))]^2$$

Non-linear generalization of PCA, can learn to project data onto non-linear manifolds.

Powerful tool for data compression.

Non-linear Autoencoders



Freeze encoder, chop off decoder and use z representation as input to simple discriminative head, can work well in some cases!

Problems:

compressed z representation potentially has complex structure, thus using this representation in downstream task can be hard

Generating new data from Autoencoder in principled way is hard

Alternative: Maximize Likelihood of Data

Maximum likelihood objective, this time without labels:

$$L(\theta, D) = \sum_{x \sim P_{data}} -\log P_\theta(X)$$

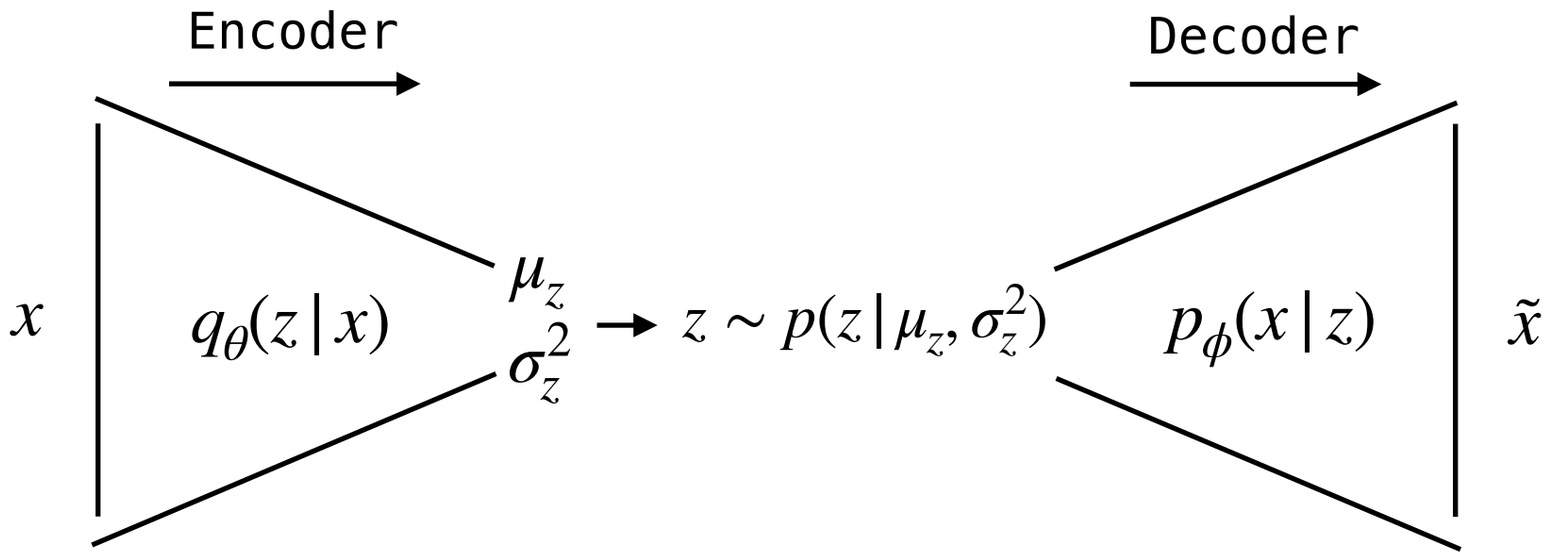
Equivalent to minimizing KL-divergence between model and data distribution:

$$D_{KL}(P_{data} \,||\, P_\theta) = \sum_{x \sim P_{data}} P_{data}(X) \log \frac{P_{data}(X)}{P_\theta(X)} = -H(X) - \mathbb{E}_{x \sim P_{data}} \left[\log P_\theta(x)\right]$$

Any divergence is zero iff: $P_{data} = P_\theta$

Implication: Incentivises to model everything about the data!

Variational AutoEncoders



Re-interpret Autoencoder as latent variable models:

$$\log p(x) = \log \int_z p(z)p(x|z)dz$$

Do variational approximation to probabilistic encoder:

$$= \log \mathbb{E}_{z\sim q_\theta(z|x)}[\frac{p(z)p(x|z)}{q_\theta(z|x)}]$$

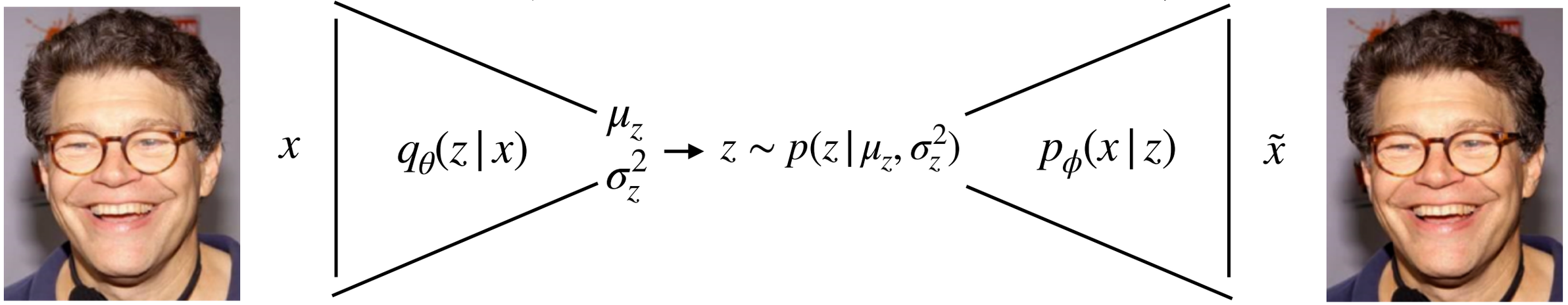
Apply Jensen's inequality:

$$\geq \mathbb{E}_{z\sim q_\theta(z|x)}[\log \frac{p(z)p(x|z)}{q_\theta(z|x)}]$$

Variational Autoencoder objective:

$$= \mathbb{E}_{z\sim q_\theta(z|x)}[p(x|z)] - D_{KL}(q_\theta(z|x)||p(z))$$

Variational AutoEncoders



Re-interpret Autoencoder as latent variable models:

$$\log p(x) = \log \int_z p(z)p(x|z)dz$$

Do variational a
probabilistic e

Equality when variational approximation matches true posterior:

$$p(z|x) = q_\theta(z|x)$$

$$= \log \mathbb{E}_{z\sim q_\theta(z|x)}[\frac{p(z)p(x|z)}{q_\theta(z|x)}]$$

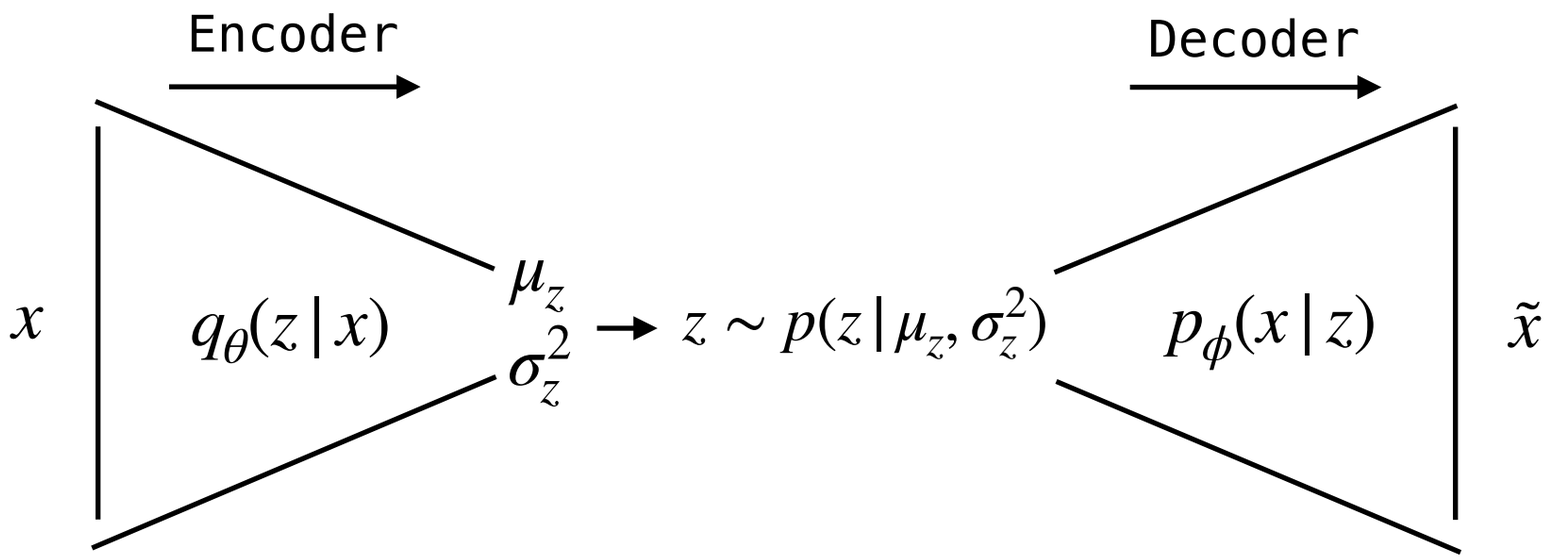
Apply Jensen's

$$\geq \mathbb{E}_{z\sim q_\theta(z|x)}[\log \frac{p(z)p(x|z)}{q_\theta(z|x)}]$$

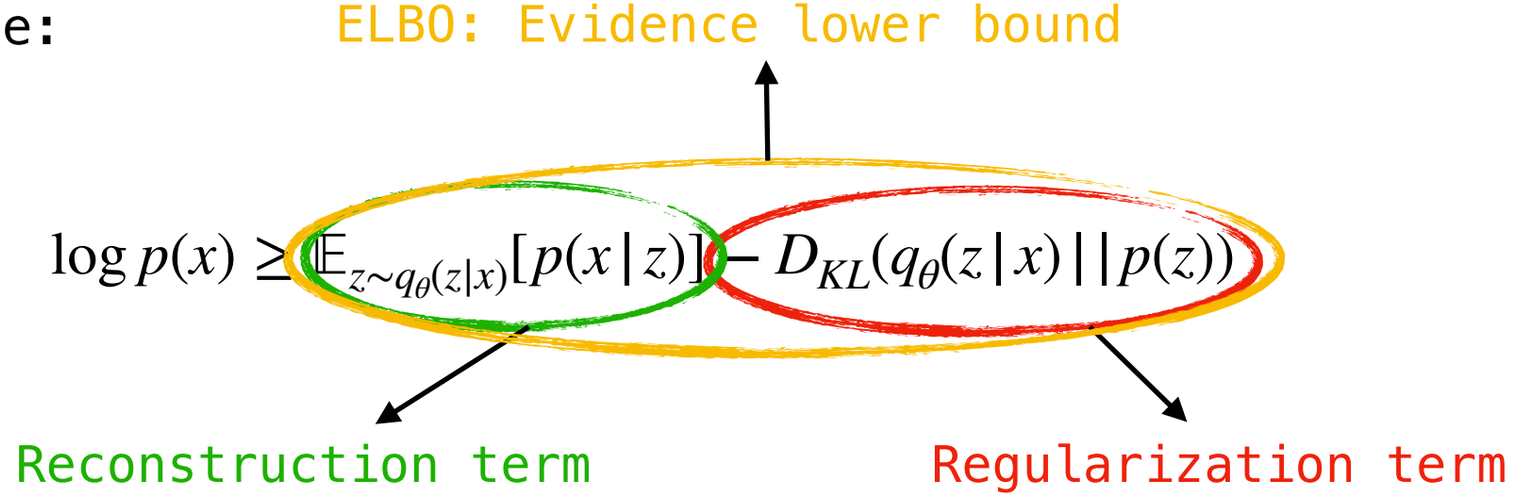
Variational Autoencoder objective:

$$= \mathbb{E}_{z\sim q_\theta(z|x)}[p(x|z)] - D_{KL}(q_\theta(z|x)||p(z))$$

Variational AutoEncoders



Final objective:



Lower bound on data likelihood, which we can maximize in training!

Concept is very general and many choices of encoder, decoder and prior are possible

Variational **A**uto**E**ncoder Applications

Once VAE is trained we can generate new data, e.g. by interpolating in latent space, here trained on celebrity images.

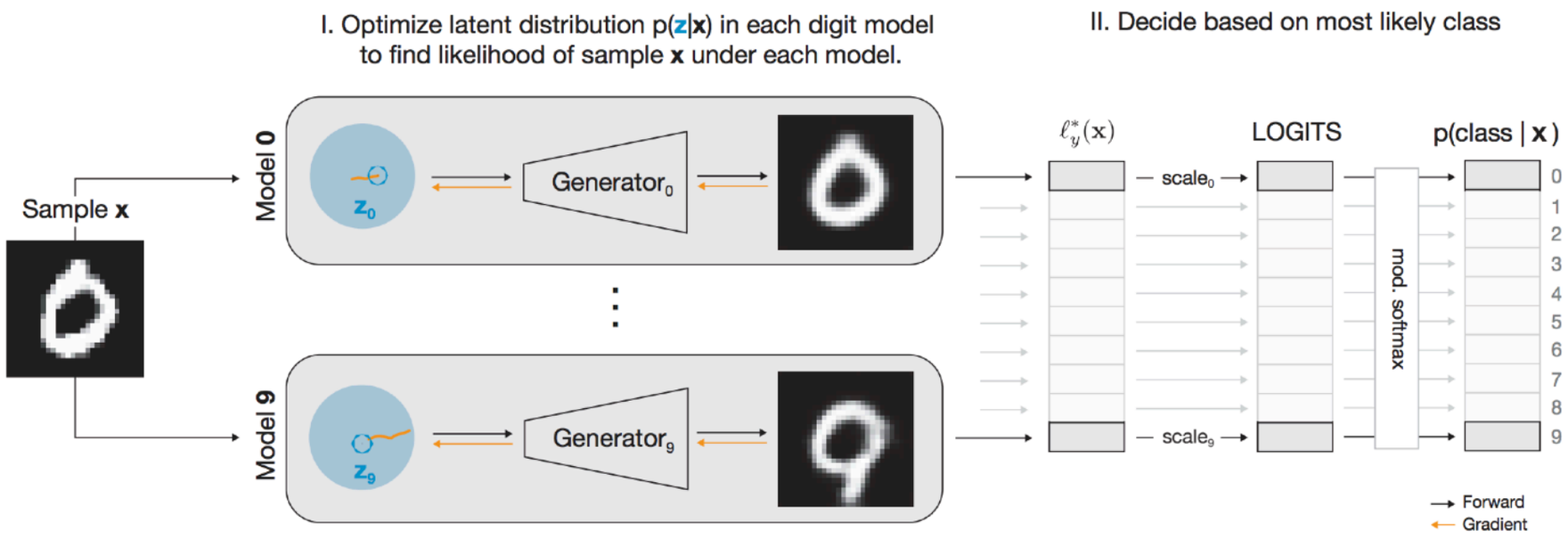
[“Sampling Generative Networks”; White, 2016]

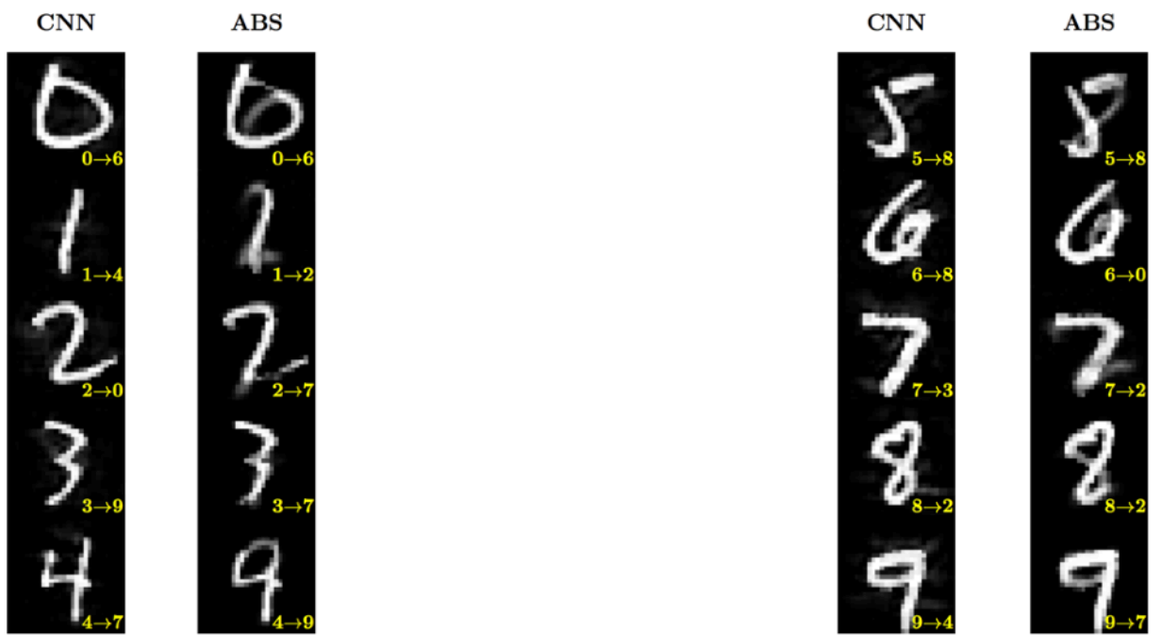
Variational **A**uto**E**ncoder Applications

VQ-VAE extension, adds discrete encoder and learned prior into mix.

[“Generating Diverse High-Fidelity Images with VQ-VAE-2”; Razavi et al., 2019]

Variational AutoEncoder Applications

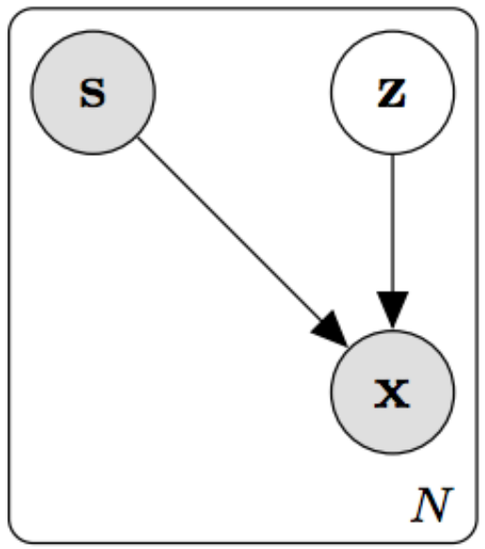


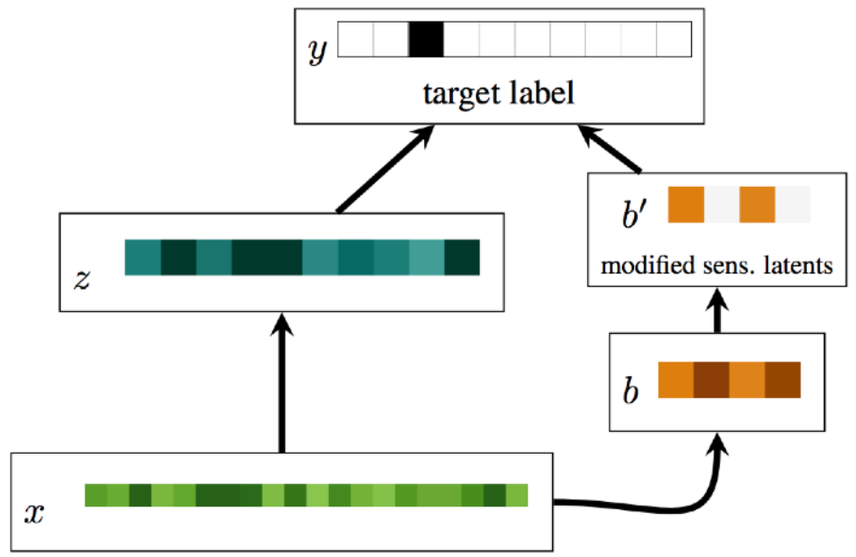


SOTA in Adversarially robust classification on MNIST

[“Towards the first robust models on MNIST”; Schott et al., 2018]

Variational AutoEncoder Applications





Promising results in Fair representation learning.

One major goal: Learn representations invariant to changes of “sensitive attributes” in inputs. Can easily be phrased as constrained on latent space!

[“Variational Fair Autoencoder”; Louizos et al., 2016,
“Flexibly Fair Representation Learning by Disentanglement”; Creager et al., 2019]

Alternative to Lower Bound: Exact Likelihood

Use chain rule of probability and break high-dimensional modeling problem into 1-dimensional conditional density modeling problems:

Time series: $$p(x) = p(x_1, x_2, \ldots, x_N) = \prod_{i=1}^{N} p(x_i \,|\, x_{i-1}, x_{i-2}, \ldots, x_1)$$

Images: $$p(x) = \prod_i \prod_j p(x_{i,j,k} \,|\, x_{<})$$

Video: $$p(x) = \prod_i \prod_j \prod_k p(x_{i,j,k} \,|\, x_{<})$$

Easy to extend to various data domains and dimensionalities

Typically called: “Autoregressive Models”

Autoregressive Models

Bigger models are always better — $p(x_1)$

Bigger models are always better — $p(x_2 \mid x_1)$

Bigger models are always better — $p(x_3 \mid x_2, x_1)$

Bigger models are always better — $p(x_4 \mid x_3, x_2, x_1)$

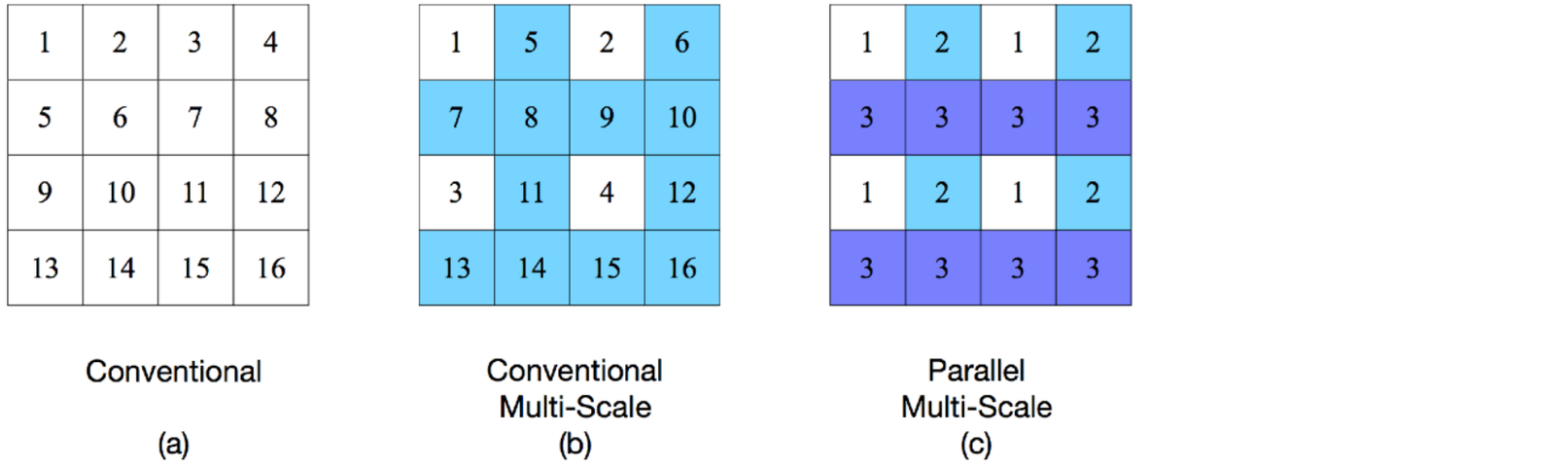
Bigger models are always better — $p(x_5 \mid x_4, x_3, x_2, x_1)$

NLL Objective: $$L(\theta, D) = \sum_{x \sim P_{data}} \sum_{i=1}^{N} -\log p_\theta(x_i \mid x_{<i})$$

In practice, probability is discrete softmax output, similar to training classifier.

Conditioning is done via network state, e.g. RNNs, LSTMs, masked convolutions or Transformers.

Autoregressive Models

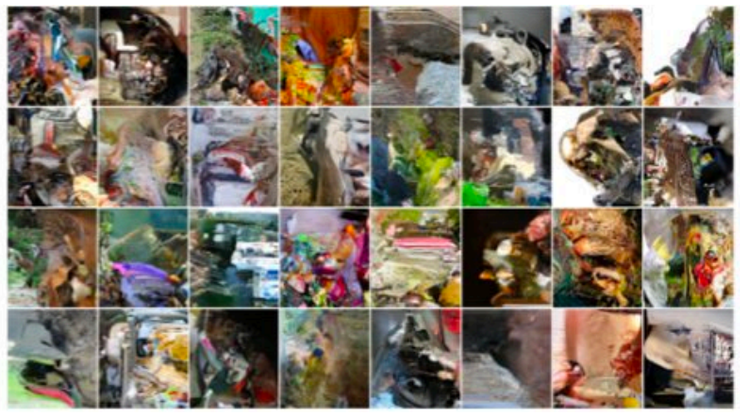


One caveat: ordering of factorization is important.

Recent progress mostly based on finding better orderings for data!

["Generating high fidelity images with subscale pixel networks and multidimensional upscaling"; Menick et al., 2018]

Autoregressive Models



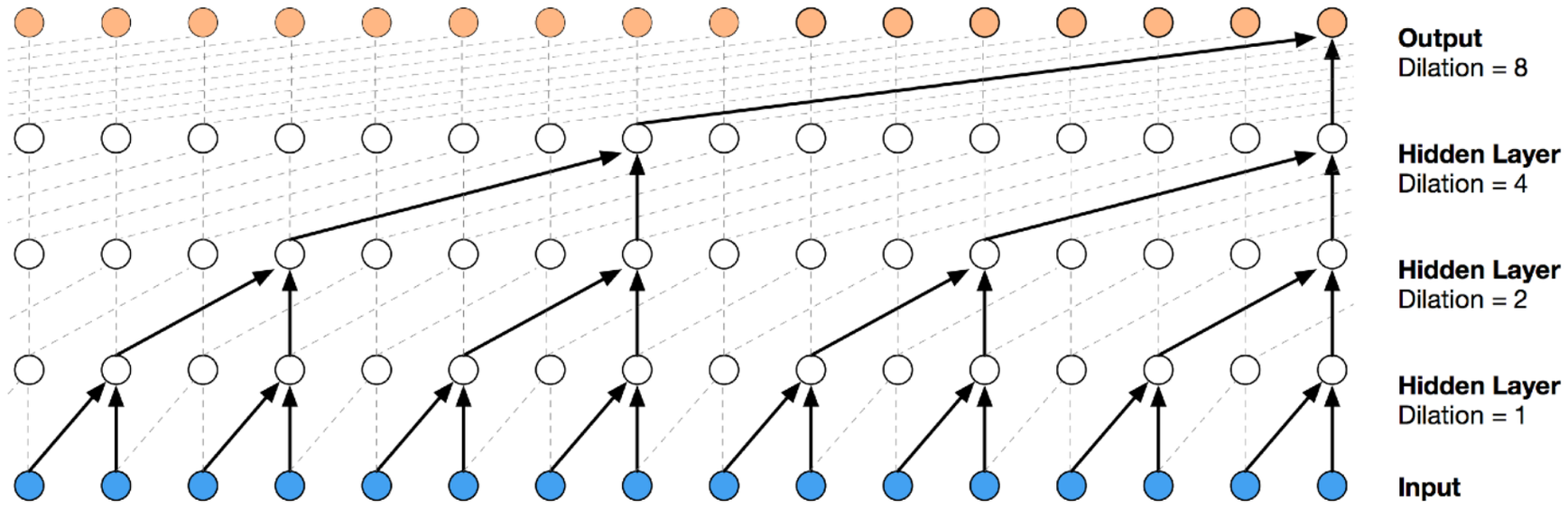
PixelRNN (2016) ImageNet 64 x 64

SPN (2018) ImageNet 128 x 128

Main difference between PixelRNN and SPN: dimension ordering, suitable for data domain (+ improved models)!

[“Generating high fidelity images with subscale pixel networks and multidimensional upscaling”; Menick et al., 2018]

Autoregressive Models



WaveNet uses dilated causal convolutions to model audio autoregressively, large receptive field important due to high temporal resolution of raw audio!

Autoregressive models have been applied to many more modalities like language, tabular data …

[“WaveNet: A Generative Model for Raw Audio”; Van den Oord et al., 2016]

Autoregressive Models

Summary

Autoregressive models enable tractable and expressive modeling of arbitrary data distributions via chain rule of probability

No mode collapse, and generalization is easy to quantify

State-of-the-art for density estimation in various domains

Downsides:

Ordering of factorization matters a lot!

Training can be parallelized, but sampling is slow

Architectures are very different from SOTA discriminative models, potentially upper-bounding performance on downstream tasks

Unifying Discriminative and Generative Models

Autoregressive models have very different structure than the state-of-the-art in discriminative tasks ***–> i.e. ResNet-like architectures***

Why does this matter?

If model does not work well in discriminative setting, probably won't provide gain when pre-trained unsupervised

There is a family of generative models which is quite similar to the best supervised models: Normalizing Flows!

Simple and elegant way to train with exact maximum likelihood via change-of-variable formula:

$$\log P_X(x) = \log P_Z(z) + \log |\det J_F(x)|$$

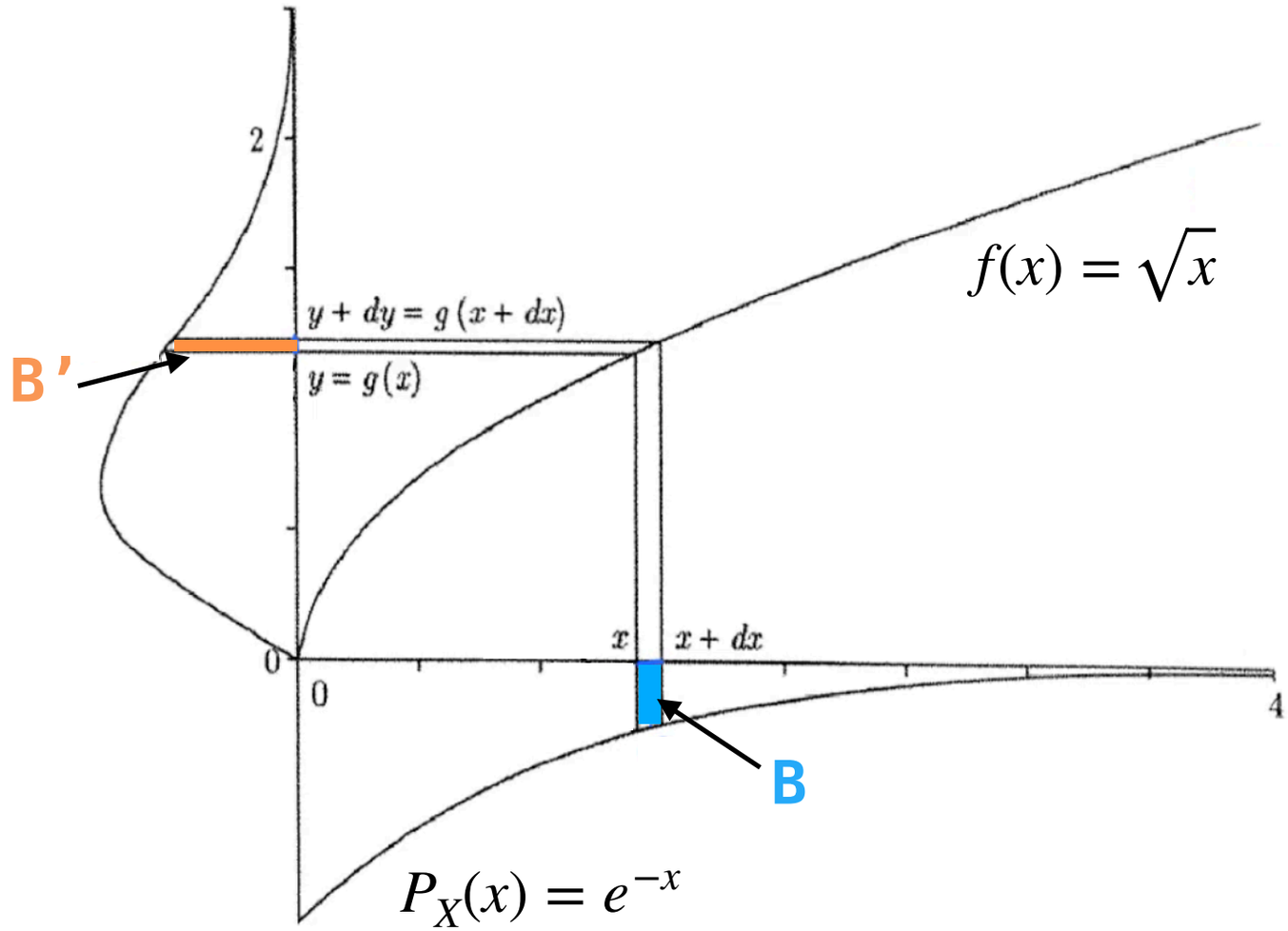
Change of Variable Formula Intuition

$P_X(x)$ is transformed by bijective function $f(x)$, result is $P_Y(y)$.

Intervals are “stretched”. However, we know:

$$P(Y \in B') = P(X \in B)$$

Change of variable formula relates two areas under graph.



Change of Variable Formula Intuition

$P_X(x)$ is transformed by bijective function $f(x)$, result is $P_Y(y)$.

Intervals are “stretched”. However, we know:

$$P(Y \in B') = P(X \in B)$$

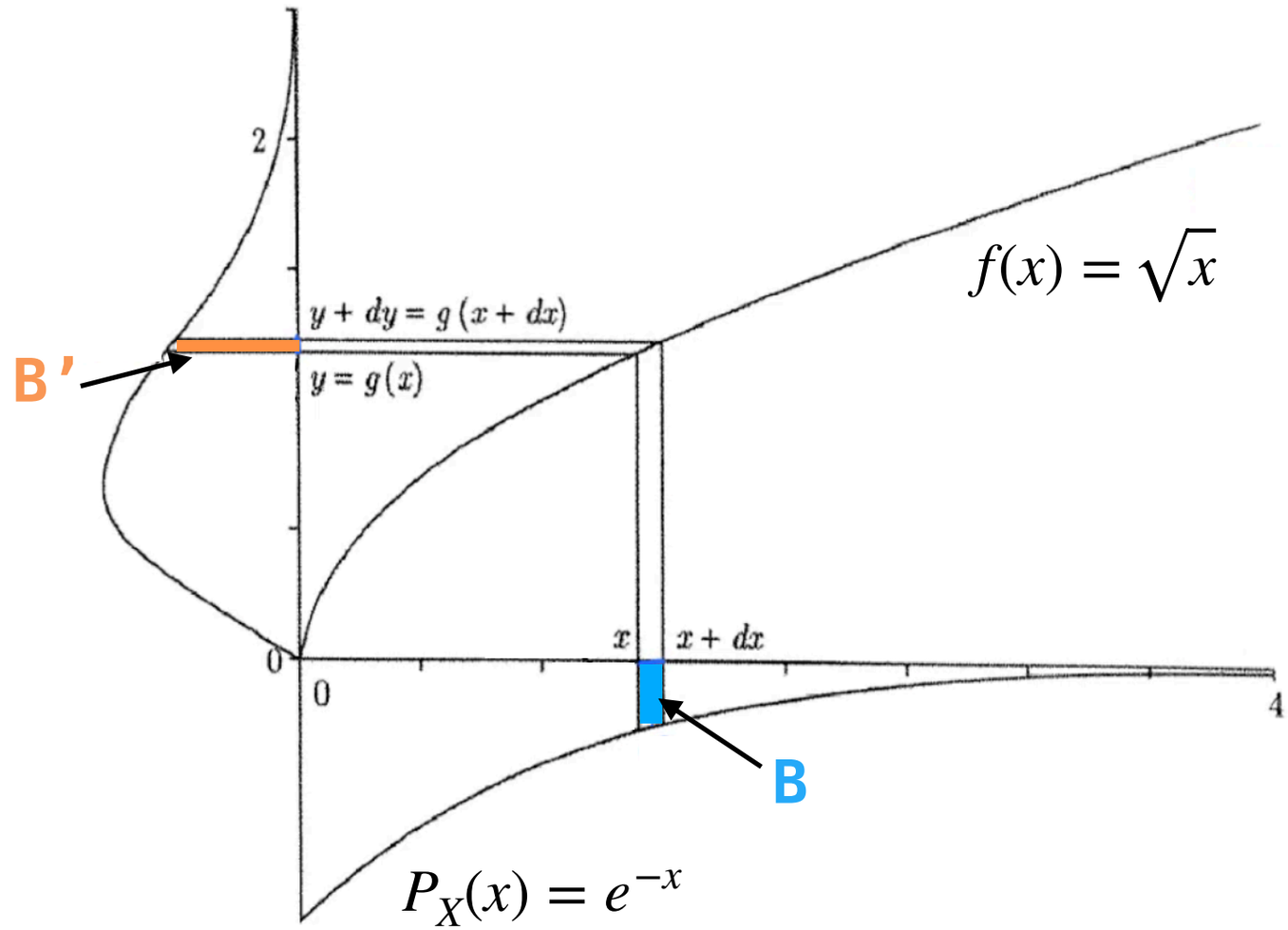
Change of variable formula relates two areas under graph.

By considering differential volume element we see:

$$P_Y(y)dy = P_X(x)dx$$

$$P_Y(y) = P_X(x)\left|\frac{dx}{dy}\right|$$

Without knowledge of functional form of $P_Y(y)$ we can evaluate its density!



Change of Variable Formula Intuition

$P_X(x)$ is transformed by bijective function $f(x)$, result is $P_Y(y)$.

Intervals are “stretched”. However, we know:

$$P(Y \in B') = P(X \in B)$$

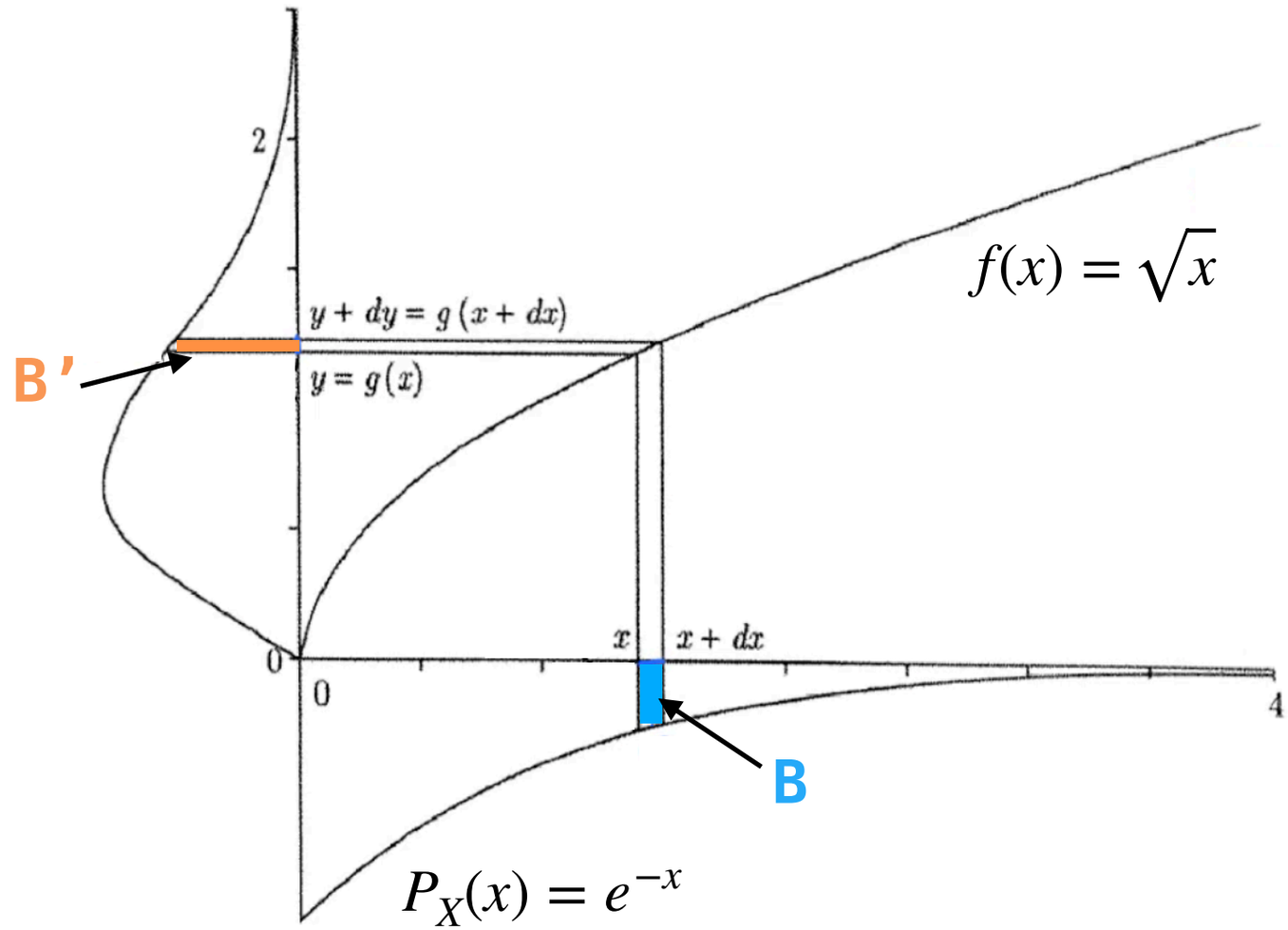
Change of variable formula relates two areas under graph.

By considering differential volume element we see:

$$P_Y(y)dy = P_X(x)dx$$

$$P_Y(y) = P_X(x)\left|\frac{dx}{dy}\right|$$

Without knowledge of functional form of $P_Y(y)$ we can evaluate its density!



Generalizing to higher dimensions and taking log:

$$\log p_X(x) = \log p_Y(y) + \log \left| det\, J_f(x) \right|$$

Training objective of normalizing flows!

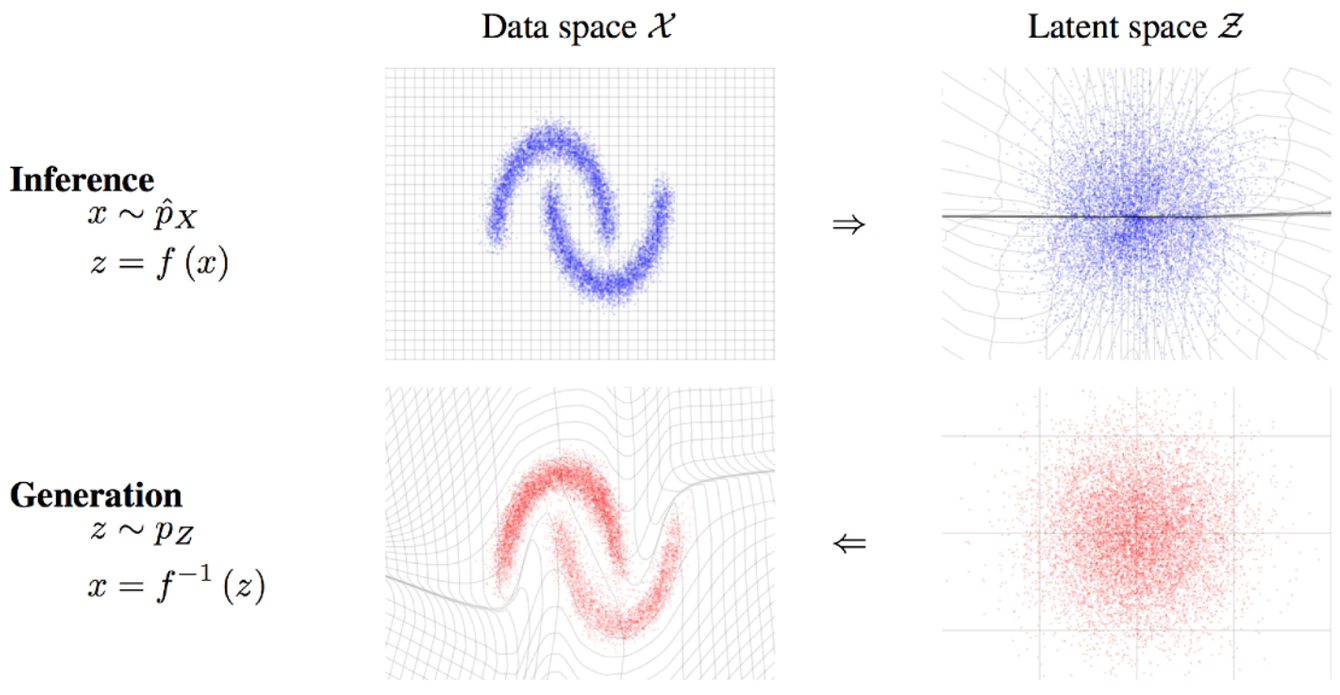
Invertible Neural Nets and Normalizing Flows

Now we know how to relate densities transformed by invertible functions to one another.

Normalizing Flow idea: specify *simple base distribution in output space* and compute *complex data-likelihood in input space via change-of-variable formula.*

Problems:

1) We need expressive bijective deep networks
2) Log determinant of Jacobian of this network needs to be easy to compute
3) Sampling from model requires tractable inverse

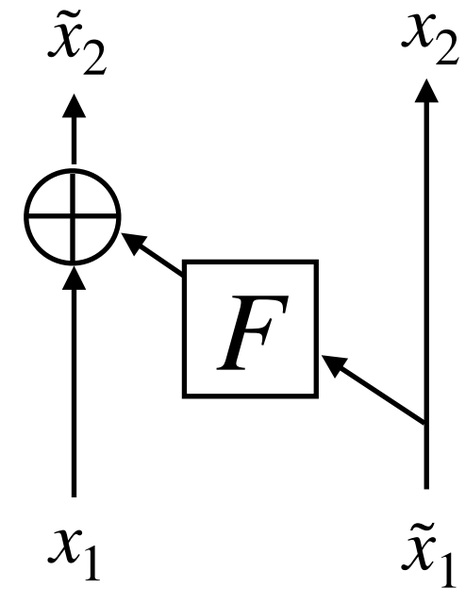


Nonlinear **I**ndependent **C**omponent **A**nalysis

NICE builds invertible network as composition of volume conserving invertible blocks.

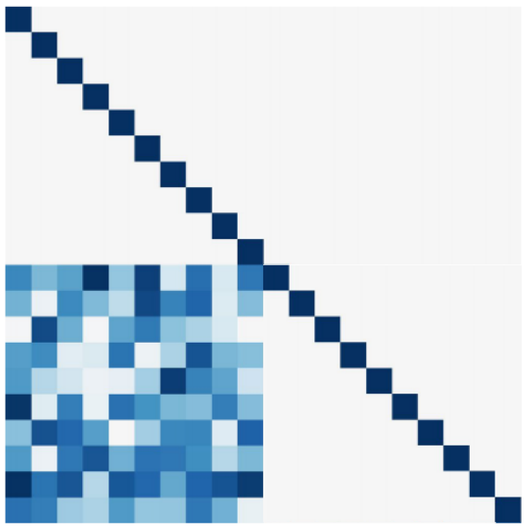
Recall ResNet block:

$$x_2 = x_1 + F_\theta(x_1)$$



“Coupling Block” differs in dimension splitting:

$$x_2 = x_1$$
$$\tilde{x}_2 = x_1 + F_\theta(\tilde{x}_1)$$

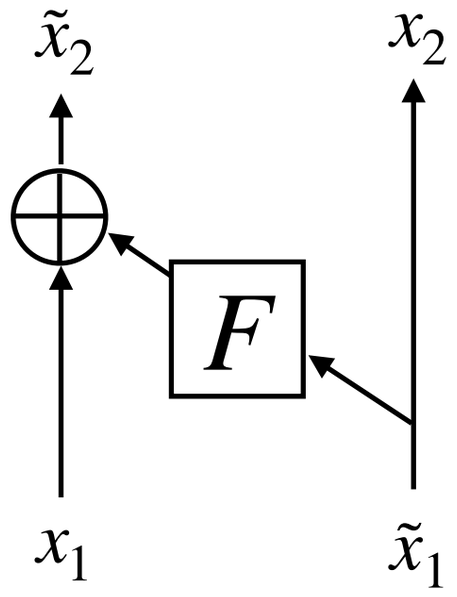


Nonlinear **I**ndependent **C**omponent **A**nalysis

NICE builds invertible network as composition of volume conserving invertible blocks.

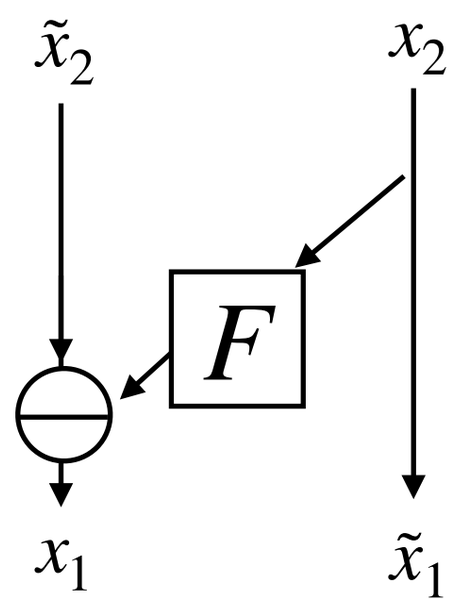
Recall ResNet block:

$$x_2 = x_1 + F_\theta(x_1)$$



“Coupling Block” differs in dimension splitting:

$$x_2 = x_1$$
$$\tilde{x}_2 = x_1 + F_\theta(\tilde{x}_1)$$



Inverse is simple and efficient:

$$\tilde{x}_1 = x_2$$
$$x_1 = \tilde{x}_2 - F_\theta(\tilde{x}_1)$$

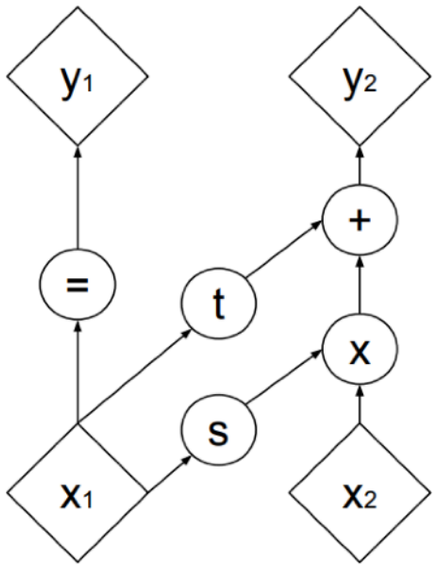
Jacobian is triangular with ones on diagonal, NICE is volume conserving:

$$\log|detJ_F(x)| = 0$$

[“Nonlinear Independent Component Analysis”; Dinh et al., 2014]

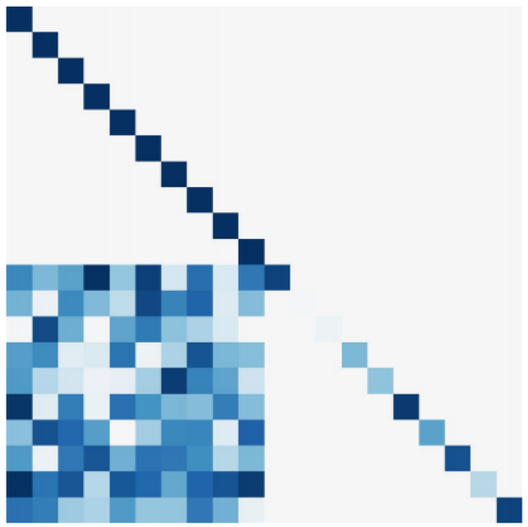
Real-NVP: Non-volume Conserving NICE

Main idea is same as NICE, but additive coupling block is extended to affine transformation and multi-scale architecture is introduced.



$$\begin{cases} y_{1:d} &= x_{1:d} \\ y_{d+1:D} &= x_{d+1:D} \odot \exp\left(s(x_{1:d})\right) + t(x_{1:d}) \end{cases}$$

$$\Leftrightarrow \begin{cases} x_{1:d} &= y_{1:d} \\ x_{d+1:D} &= \left(y_{d+1:D} - t(y_{1:d})\right) \odot \exp\left(-s(y_{1:d})\right) \end{cases}$$



Just like NICE, model is trained by minimizing negative data log-likelihood, where log det term is sum over log of diagonal entries:

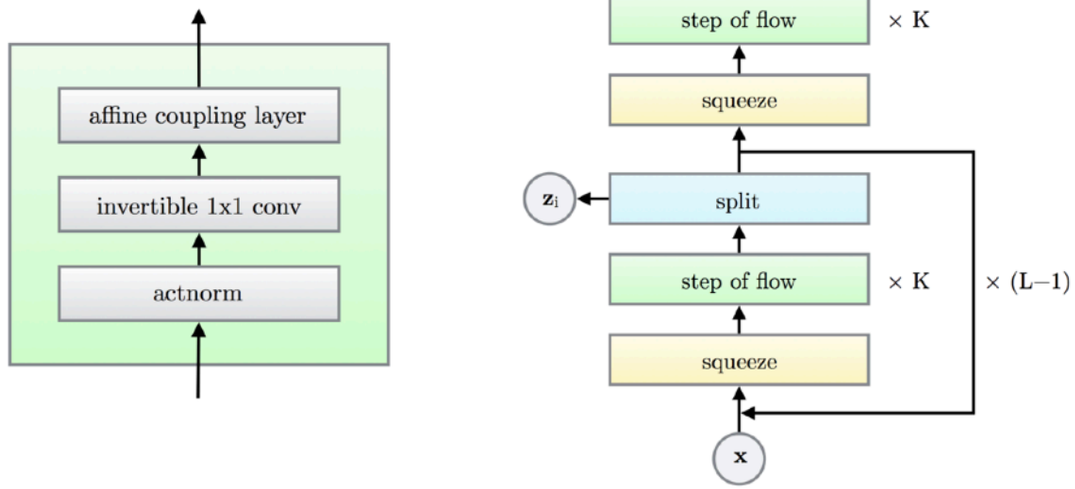
$$L(\theta, D) = \sum_{x \sim P_{data}} -\log p_Z(f_\theta(x)) - \log \left| det J_f(x) \right|$$

["Density estimation using Real NVP"; Dinh et al., 2016]

GLOW

Glow improved upon Real-NVP in multiple ways:

1) Learn dimension splitting with 1x1 convolutions
2) Removes batch dependency due to Batchnorm
3) Scales models up a lot



(a) One step of our flow. (b) Multi-scale architecture (Dinh et al., 2016).

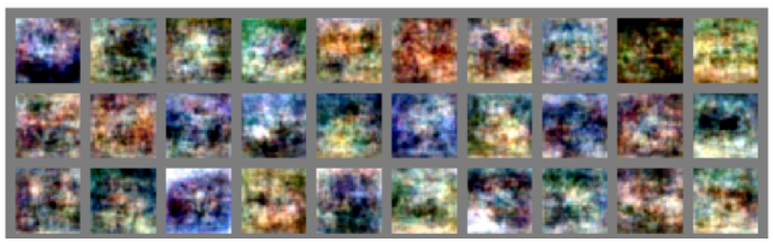
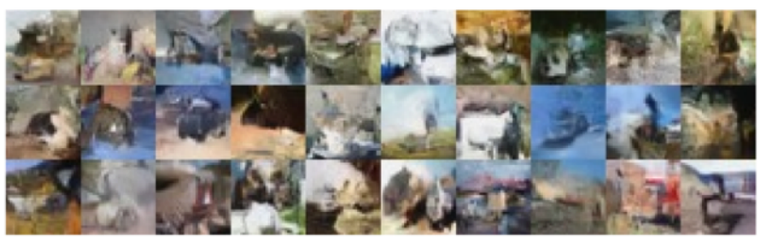
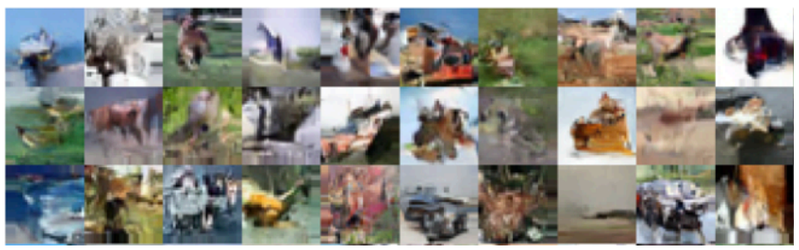
Table 2: Best results in bits per dimension of our model compared to RealNVP.

Model	CIFAR-10	ImageNet 32x32	ImageNet 64x64	LSUN (bedroom)	LSUN (tower)	LSUN (church outdoor)
RealNVP	3.49	4.28	3.98	2.72	2.81	3.08
Glow	**3.35**	**4.09**	**3.81**	**2.38**	**2.46**	**2.67**

Results start to close gap to autoregressive models and also showed that stability as well as hand-designed dimension splitting are major bottlenecks in flow models.

["Glow: Generative Flow with Invertible 1x1 Convolutions"; Kingma et al., 2018]

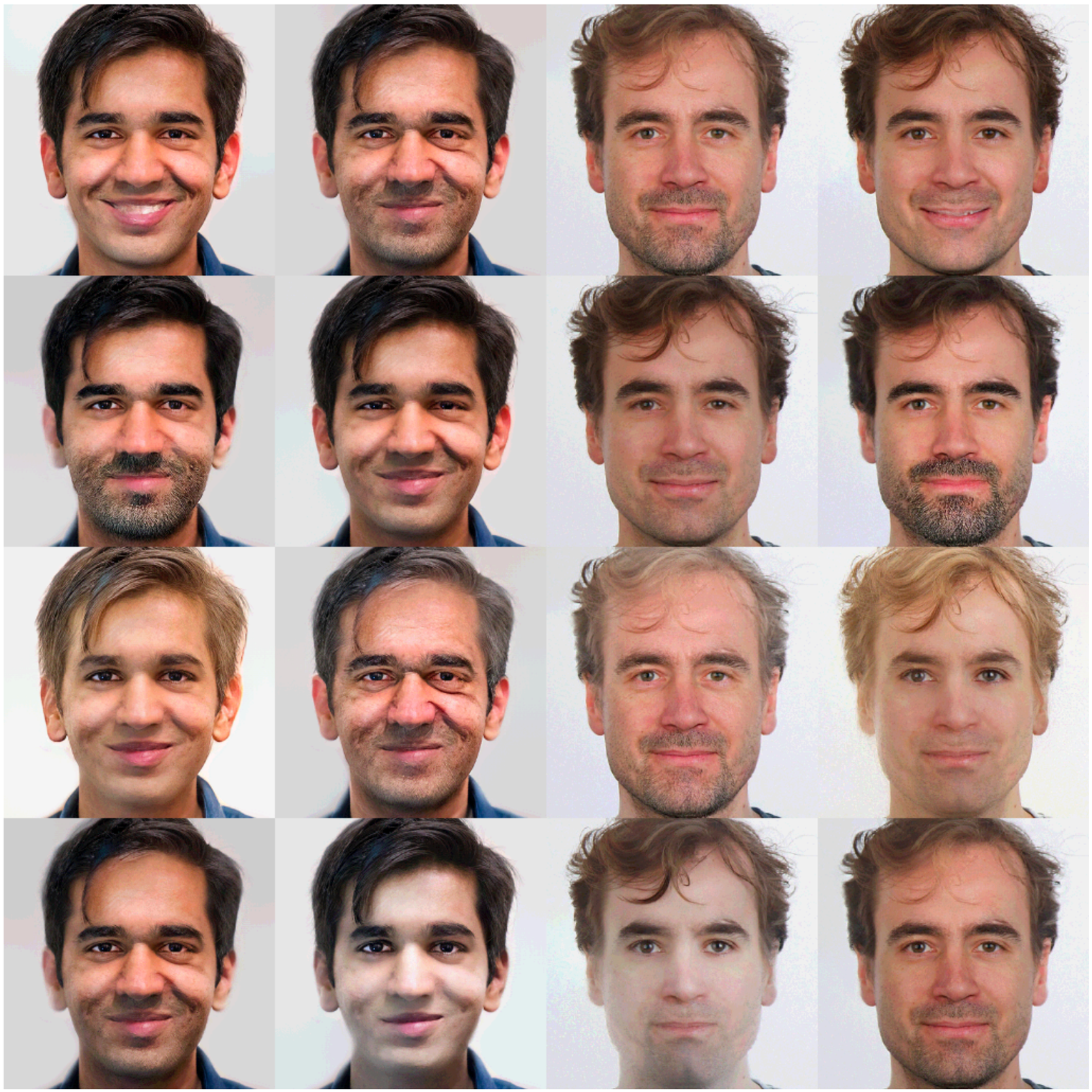
Progress over Years

NICE	Real-NVP	GLOW
		
CIFAR10 samples		
4.48	**3.49**	**3.35**
CIFAR10 negative log-likelihoods		

GLOW Samples

[“Glow: Generative Flow with Invertible 1x1 Convolutions”; Kingma et al., 2018]

GLOW Semantic Manipulations



[“Glow: Generative Flow with Invertible 1x1 Convolutions”; Kingma et al., 2018]

Taking it Further: Invertible Residual Networks

Can we guarantee invertibility in SOTA non-invertible networks?

-> i.e. without dimension splitting

Remarkable similarity between ResNets and Euler's method for ODE initial value problems:

$$x_{t+1} = x_t + g_{\theta_t}(x_t) \qquad \text{ResNet Update}$$

$$x_{t+1} = x_t + hf_{\theta_t}(x_t) \qquad \text{Euler Update}$$

Dynamics backward in time are given by implicit backward Euler discretization:

$$x_t = x_{t+1} - g_{\theta_t}(x_t)$$

$$x_t = x_{t+1} - hf_{\theta_t}(x_t)$$

["Invertible Residual Networks"; Berhmann et al., 2019]

Invertible Residual Networks

Compute inverse via fixed-point iteration:

Inverse:

$$x^0 = y$$

$$x^{i+1} := y - g(x^i)$$

Sufficient condition for invertibility is contraction:

$$Lip(g) < 1$$

Guarantees convergence of iteration due to Banach fixed point theorem!

[“Invertible Residual Networks”; Berhmann et al., 2019]

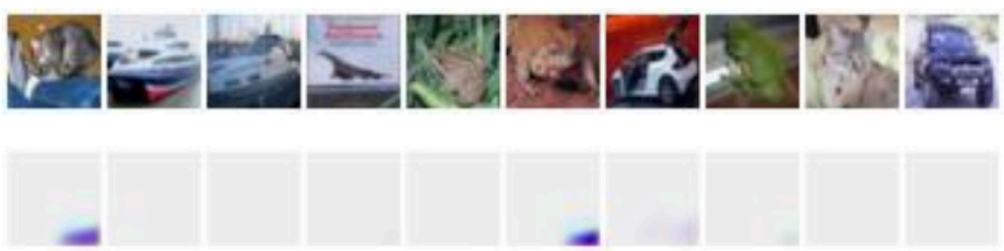
Invertible Residual Networks

Original:

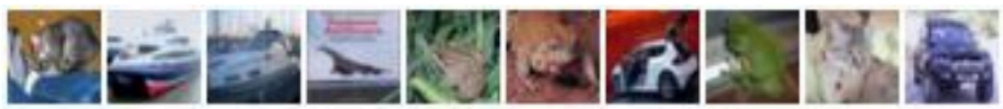
Inverse | Unconstrained:

Inverse | Lip(g) < 1:

Original:



Inverse | Unconstrained:



Inverse | Lip(g) < 1:

MNIST even invertible without constraint!

Performance not significantly affected in all cases

[“Invertible Residual Networks”; Berhmann et al., 2019]

Invertible Neural Nets and Normalizing Flows

Affine Glow 1×1 Conv	Additive Glow Reverse	i-ResNet Glow-Style	i-ResNet 164
12.63	12.36	8.03	6.69

Classification accuracy Cifar-10

Every advance in terms of density modeling performance reduced classification accuracy

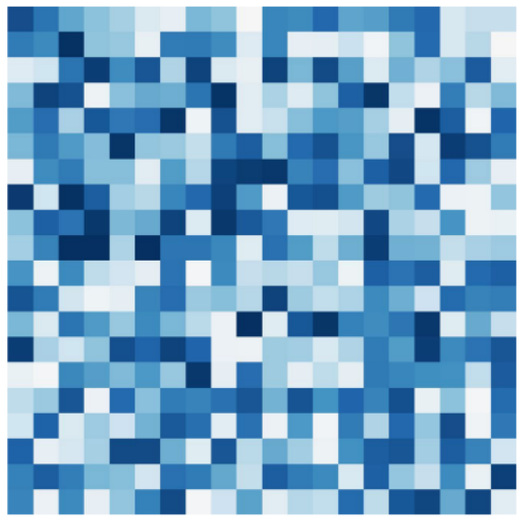
However, invertible ResNets do not suffer from this problem, they don’t need dimension partitioning and potentially unstable multiplicative interactions.

[“Invertible Residual Networks”; Berhmann et al., 2019]

Turning ResNet into Normalizing Flow

Need to estimate log determinant of Jacobian of ResNet block F(x):

$$\log p_x(x) = \log p_z(z) + \log |det\, J_F(x)|$$



[“Invertible Residual Networks”; Berhmann et al., 2019]

Turning ResNet into Normalizing Flow

Need to estimate log determinant of Jacobian of ResNet block F(x):

$$\log p_x(x) = \log p_z(z) + \log |det\, J_F(x)|$$

Lipschitz constraint perturbations of identity yield positive determinants:

$$\log |det\, J_F(x)| = \log det\, J_F(x)$$

Jacobian is non-singular, therefor:

$$\log det\, J_F(x) = tr(\log J_F(x))$$

[“Invertible Residual Networks”; Berhmann et al., 2019]

Turning ResNet into Normalizing Flow

Need to estimate log determinant of Jacobian of ResNet block F(x):

$$\log p_x(x) = \log p_z(z) + \log |det\, J_F(x)|$$

Lipschitz constraint perturbations of identity yield positive determinants:

$$\log |det\, J_F(x)| = \log det\, J_F(x)$$

Jacobian is non-singular, therefor:

$$\log det\, J_F(x) = tr(\log J_F(x))$$

As $F(x) = (I + J)(x)$, trace of matrix log can be estimated efficiently via power series:

$$tr(\log(I + J_g(x))) = \sum_{k=1}^{\infty} (-1)^{k+1} \frac{tr(J_g^k)}{k},$$

which converges if $||J_g||_2 < 1$.

[“Invertible Residual Networks”; Berhmann et al., 2019]

Turning ResNet into Normalizing Flow

In practice, i-ResNets compute truncation of infinite series and estimate trace with Skilling-Hutchinson trace estimator:

$$tr(\log(I + J_g(x))) = \mathbb{E}_{v\sim p}\left[\sum_{k=1}^{N}(-1)^{k+1}\frac{v^T J_g^k v}{k}\right].$$

Unfortunately, estimator is biased where bias is dependent on dimensionality of input and Lipschitz constant of network.

Hence, we need to carefully trade-off expressiveness and bias.

Solution: “Russian Roulette” estimator allows to estimate unbiased estimate of infinite series with probability 1 in finite time:

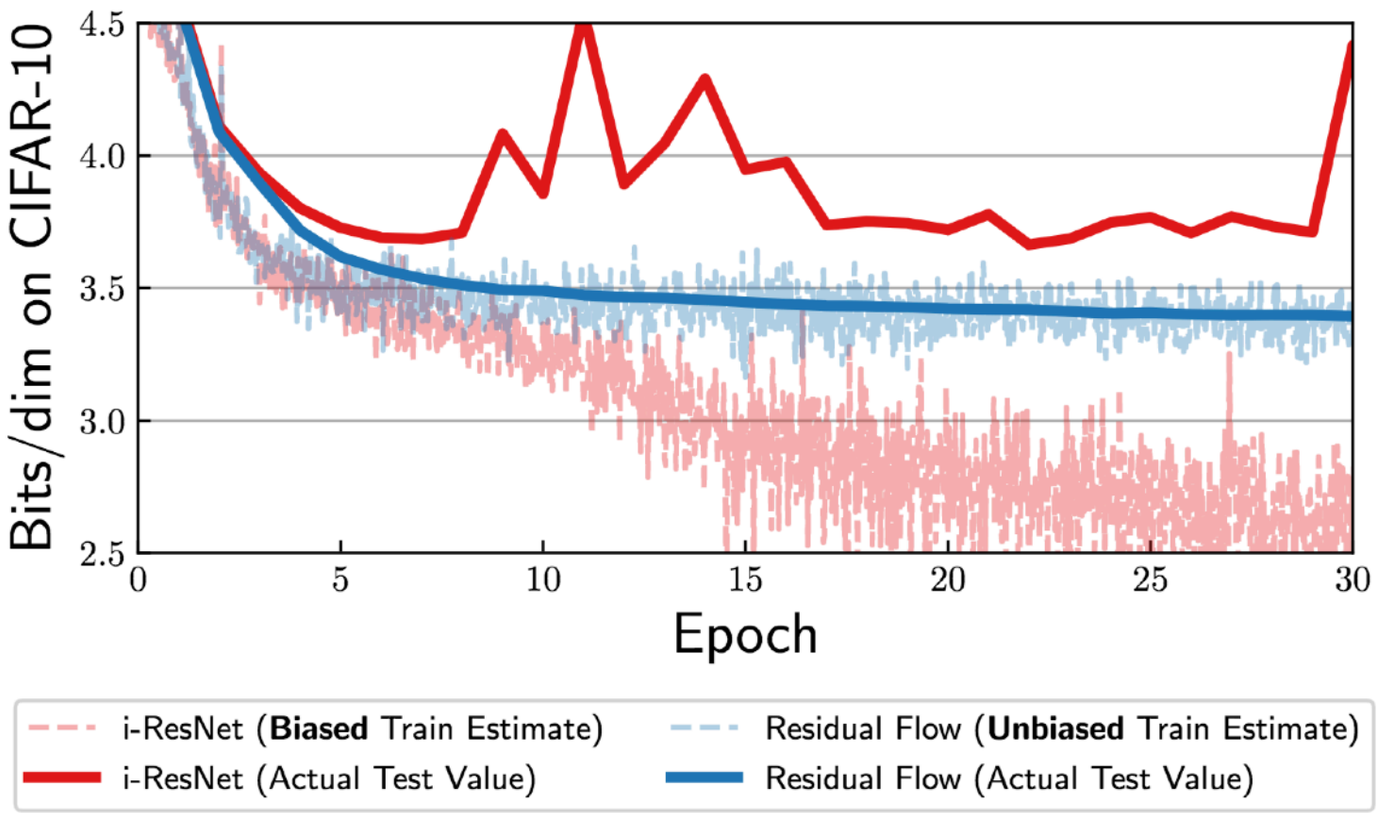
$$tr(\log(I + J_g(x))) = \mathbb{E}_{n,v\sim p}\left[\sum_{k=1}^{n}\frac{(-1)^{k+1}}{k}\frac{v^T J_g^k v}{\mathbb{P}(N \geq k)}\right].$$

Intuition: flip coin if following series term will be evaluated and appropriately weight previous terms.

[“Residual Flows for Invertible Generative Modeling”; Chen et al., 2019]

Residual Flows

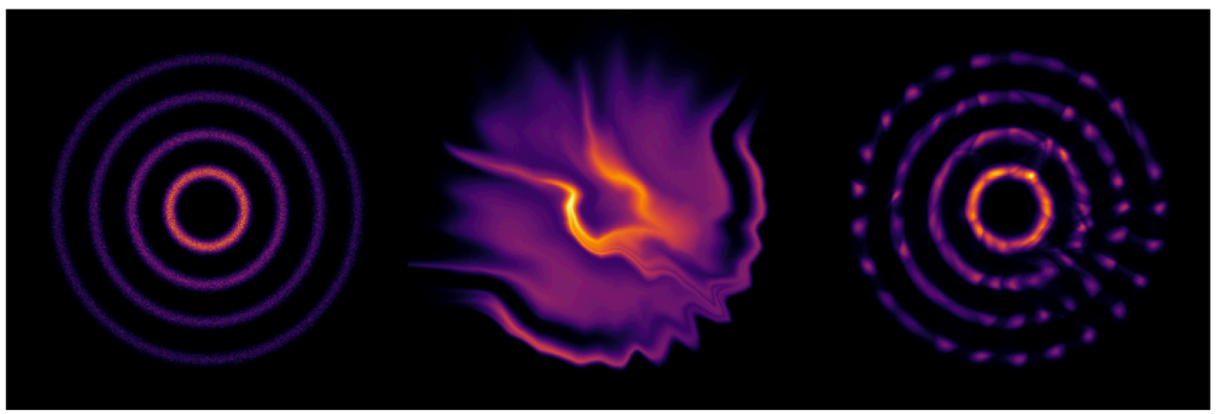
Training curves of biased and unbiased objectives:



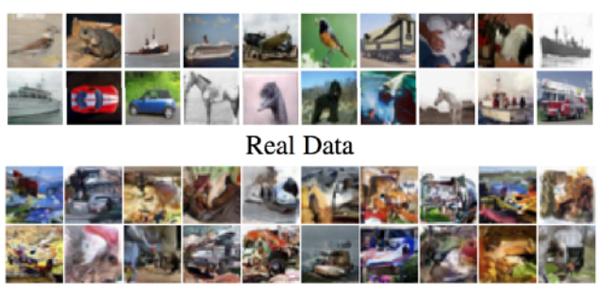
i-ResNets suffer from substantial bias when using expressive networks, whereas Residual Flows appropriately perform maximum likelihood with unbiased gradients.

["Residual Flows for Invertible Generative Modeling"; Chen et al., 2019]

Residual Flows / Invertible ResNets



Data Samples Glow i-ResNet



Real Data

PixelCNN (3.14 bits/dim)

Residual Flow (3.29 bits/dim)

Variational Dequantized Flow++ (3.08 bits/dim)

Model	MNIST	CIFAR-10	ImageNet 32×32	ImageNet 64×64
Real NVP (Dinh et al., 2017)	1.06	3.49	4.28	3.98
Glow (Kingma and Dhariwal, 2018)	1.05	3.35	4.09	3.81
FFJORD (Grathwohl et al., 2019)	0.99	3.40	—	—
Flow++ (Ho et al., 2019)	—	**3.29** (3.09)	— (3.86)	— (3.69)
i-ResNet (Behrmann et al., 2019)	1.05	3.45	—	—
Residual Flow (Ours)	**0.97**	**3.29**	**4.02**	**3.78**

Competitive or better than state-of-the-art flow models, while being based on standard ResNets!

Method of choice for all applications where competitive density estimation performance ***and*** strong discriminative downstream performance is important.

[“Residual Flows for Invertible Generative Modeling”; Chen et al., 2019]

Summary: Likelihood-based Density Modeling

3 Common Approaches

Variational Autoencoders:

Optimize tractable lower bound on data likelihood

$$\log p(x) \leq \mathbb{E}_{z \sim q}\left[\log p(x \mid z)\right] - D_{KL}(q(z \mid x) \mid\mid p(z))$$

Summary: Likelihood-based Density Modeling

3 Common Approaches

Variational Autoencoders:

Optimize tractable lower bound on data likelihood

$$\log p(x) \leq \mathbb{E}_{z\sim q}\left[\log p(x\,|\,z)\right] - D_{KL}(q(z\,|\,x)\,||\,p(z))$$

Autoregressive Models:

Apply chain rule of probability and break up N-dimensional modeling problem into N 1-dimensional conditional densities

$$p(x) = p(x_1, x_2, \ldots, x_N) = \prod_{i=1}^{N} p(x_i\,|\,x_{i-1}, x_{i-2}, \ldots, x_1)$$

Summary: Likelihood-based Density Modeling

3 Common Approaches

Variational Autoencoders:

Optimize tractable lower bound on data likelihood

$$\log p(x) \leq \mathbb{E}_{z\sim q}\left[\log p(x\,|\,z)\right] - D_{KL}(q(z\,|\,x)\,||\,p(z))$$

Autoregressive Models:

Apply chain rule of probability and break up N-dimensional modeling problem into N 1-dimensional conditional densities

$$p(x) = p(x_1, x_2, \ldots, x_N) = \prod_{i=1}^{N} p(x_i\,|\,x_{i-1}, x_{i-2}, \ldots, x_1)$$

Normalizing Flows:

Design models with tractable Jacobian determinant and match transformed data to simple base distribution with change-of-variable formula

$$\log p(x) = \log p(z) + \log\left|\det J_F(x)\right|$$

Summary: Likelihood-based Density Modeling

Variational Autoencoders:

Pro: Very flexible, few constraints on encoder / decoder / prior
Contra: Posterior collapse, often lower NLL and worse samples than exact likelihood models

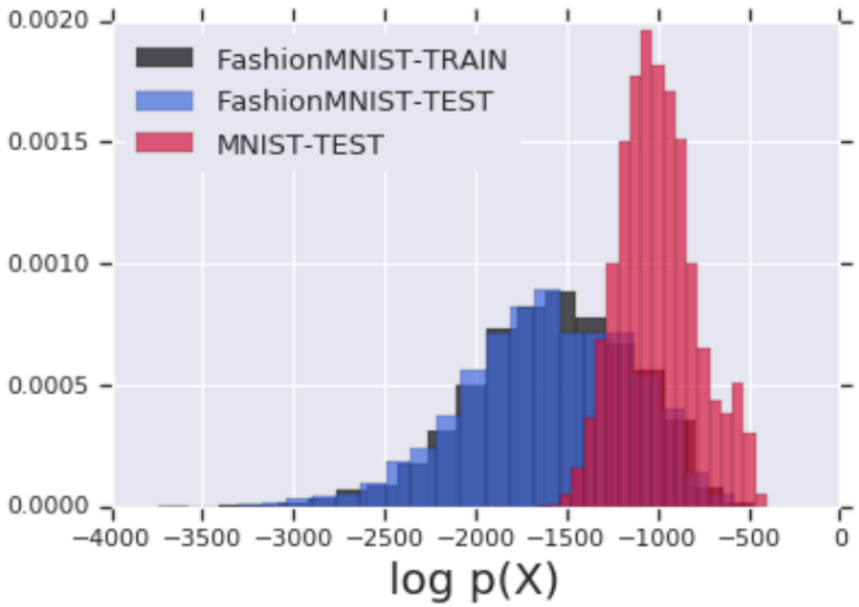
Autoregressive Models:

Pro: Simple yet powerful approach, very good performance in NLL, applicable to any data modality
Contra: Models very different from discriminative models, sampling is slow

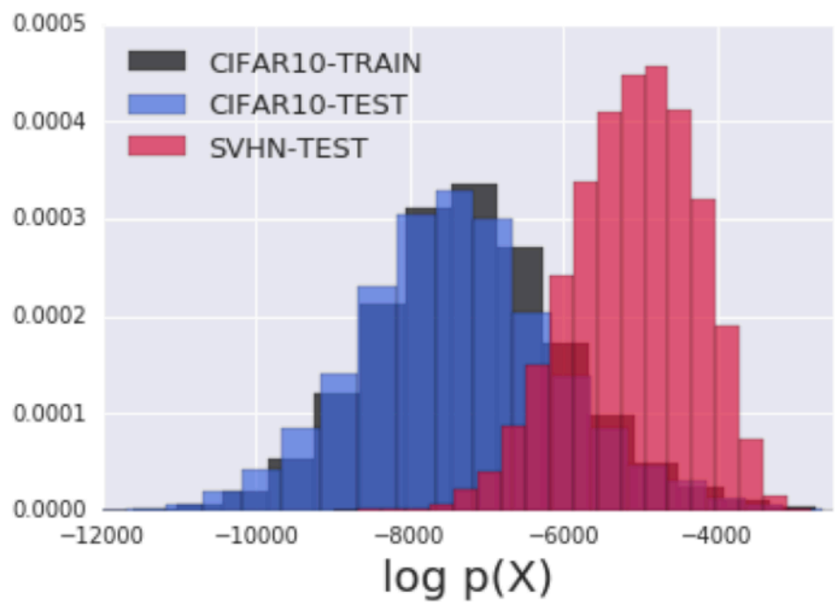
Normalizing Flows:

Pro: Simple objective, SOTA ResNets can be turned into flows with one additional constraint, excellent performance on joint discriminative and generative tasks
Contra: Need to design invertible networks with tractable Jacobian determinant

Open Issues with Likelihood-based Models



(a) Train on FashionMNIST, Test on MNIST



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

Models can assign higher likelihoods to out-of-distribution data

Trained on CIFAR10 / Fashion-MNIST, tested on SVHN / MNIST

["Do Deep Generative Models Know What They Don't Know?"; Nalisnick et al., 2018]

Open Issues with Likelihood-based Models

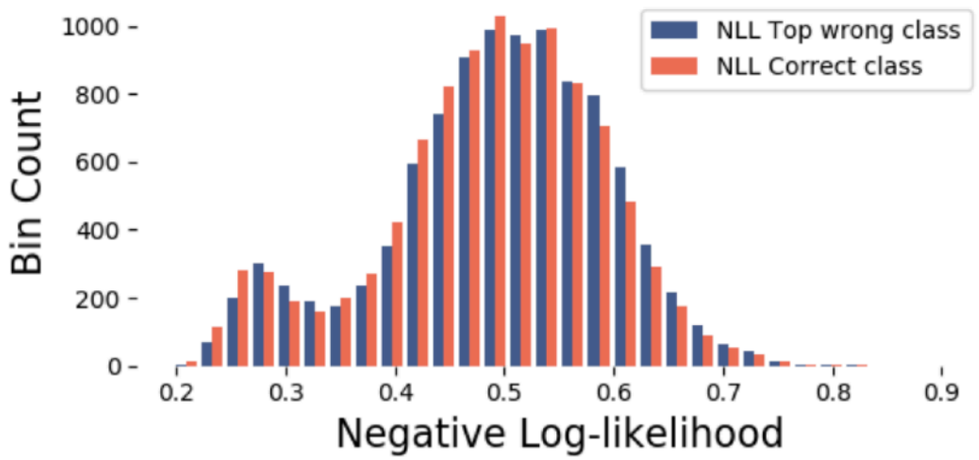
Attacking	**Classification**		**Classification and Detection**	
MNIST	Reweight	Split	Reweight	Split
$CW - L_2$	0% (100%)	1% (100%)	17% (100%)	14% (100%)
Boundary attack	43% (82%)	36% (80%)	0% (0%)	0% (0%)
CIFAR10				
$CW - L_2$	0% (97%)	0% (0%)	6% (99%)	3% (100%)
Boundary attack	67% (100%)	72% (100%)	100% (100%)	100% (100%)

Table 2: Comparison of attack detection. Percentage of successful and undetected attacks within L_2-distance of $\epsilon = 1.5$ for MNIST and $\epsilon = 33/255$ for CIFAR10 for proposed models. Number in parentheses is percentage of attacks that successfully fool the classifier, both detected and undetected.

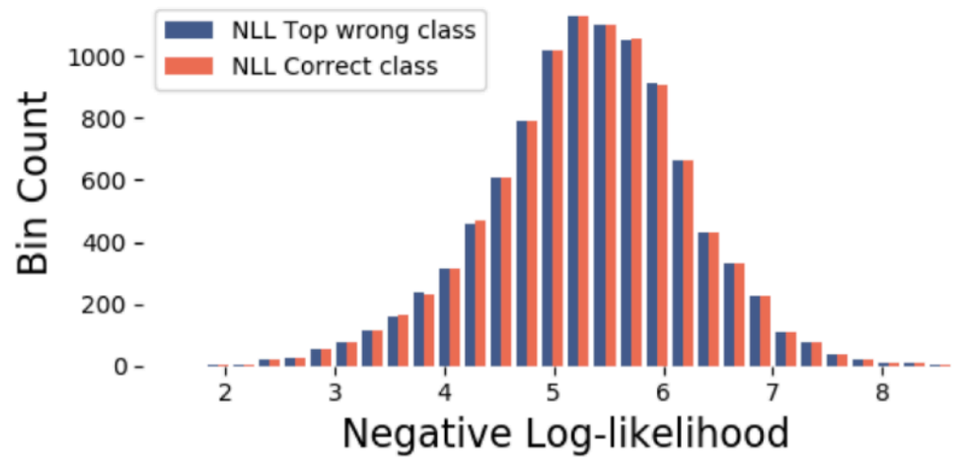
Class-conditional generative models are adversarially robust on simple buy not on complex datasets

[“Conditional Generative Models are not Robust”; Fetaya et al., 2019]

Open Issues with Likelihood-based Models



(a) MNIST



(b) CIFAR10

Figure 4: NLL for images conditioned on the correct class vs the highest probability wrong class.

Conditioning on wrong class does not count as outlier in class-conditional generative model

Strong models on both classification and log-likelihood metrics are still an open problem!

["Conditional Generative Models are not Robust"; Fetaya et al., 2019]

Summary: Likelihood-based Density Modeling

Overall pros of likelihood-based generative modeling:

Principled and powerful objective to train and quantify progress!

Likelihood allows to measure generalization on test set (memorization of training set is not enough to do well here)

Normalized densities for downstream tasks like e.g. outlier detection, robust classification with class-conditional generative models etc.

Much work to be done still!

UNIVERSITY OF TORONTO

Thank you!

Reach out: j.jacobsen@vectorinstitute.ai
Twitter: @jh_jacobsen