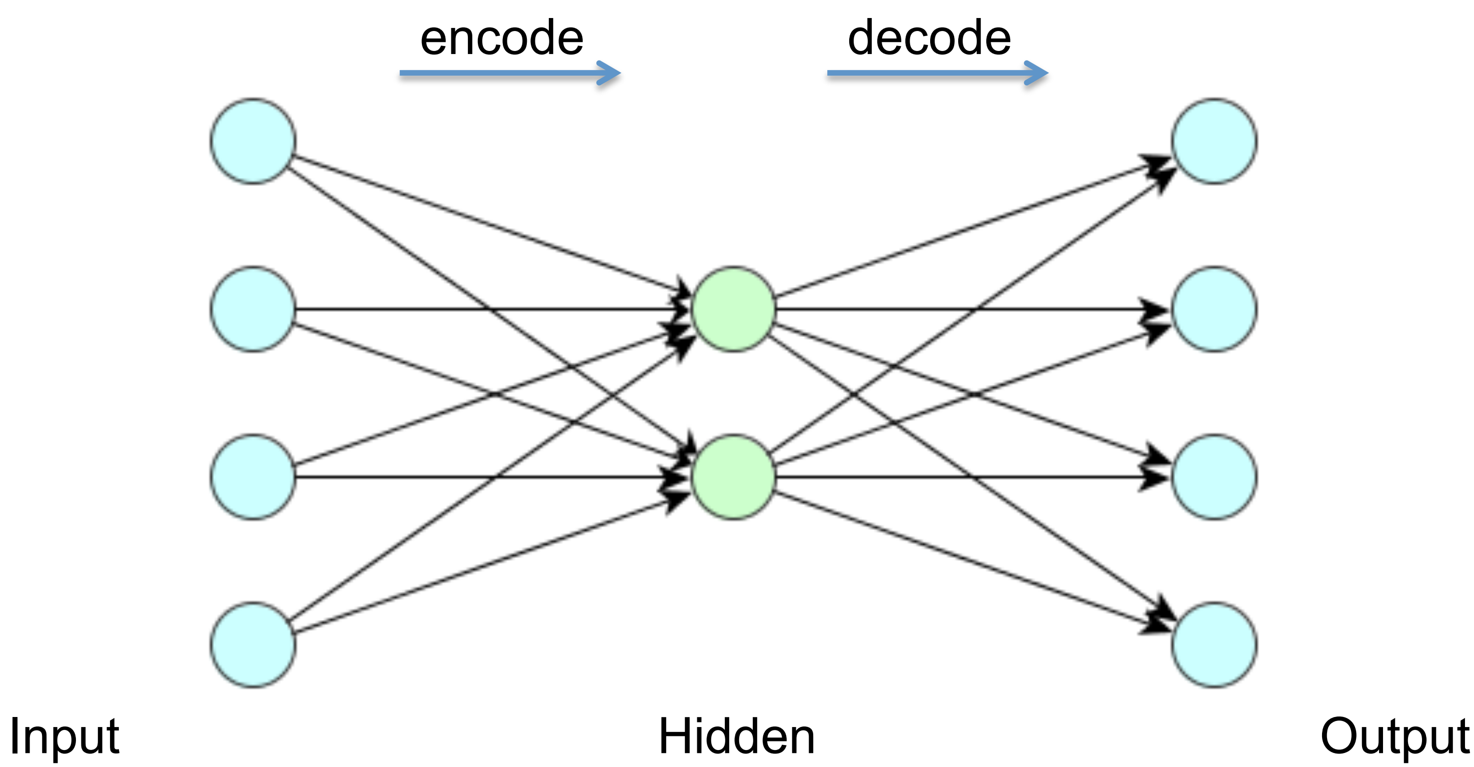
Lecture 3

Variational Auto-encoders

INTRODUCTION VARIATIONAL AUTO-ENCODERS

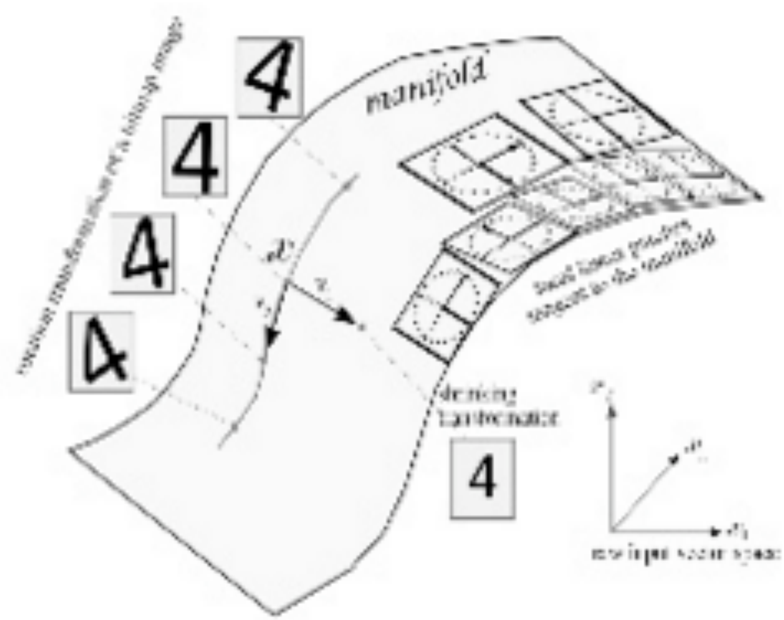
In this talk I will in some detail describe the paper of Kingma and Welling. "*Auto-Encoding Variational Bayes*, *International Conference on Learning Representations." ICLR,* 2014. arXiv:1312.6114 [stat.ML].

INTRODUCTION VARIATIONAL AUTO-ENCODERS



MANIFOLD HYPOTHESIS

- X high dimensional vector
- Data is concentrated around a low dimensional manifold



- Hope finding a representation Z of that manifold.

MANIFOLD HYPOTHESIS

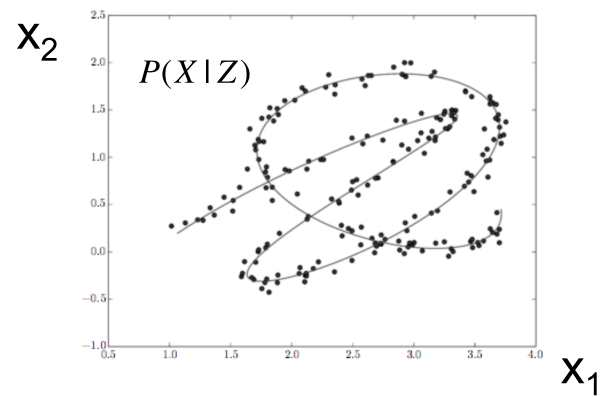
1D

Low Dimensional representation a line

z_1

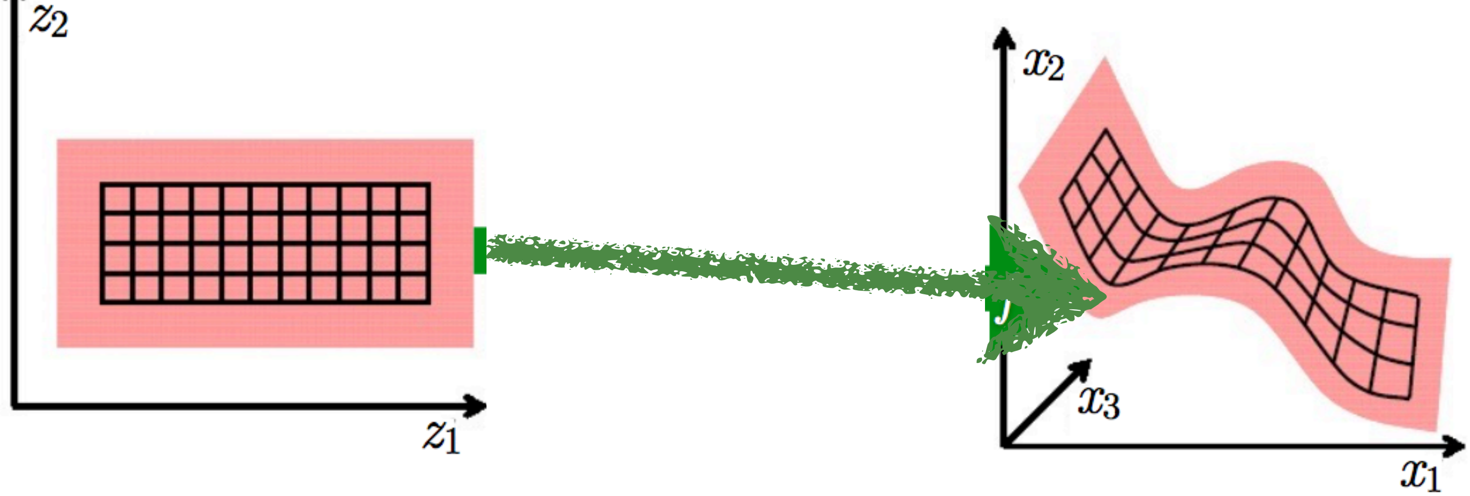
2D

High Dimensional (number of pixels)



2D

3D

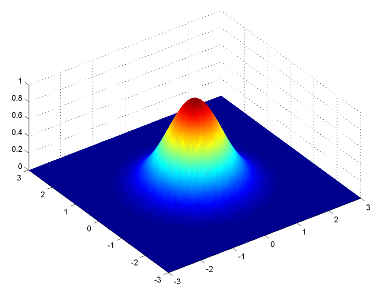


PRINCIPLE IDEA ENCODER NETWORK

- We have a set of N-observations (e.g. images) $\{x^{(1)},x^{(2)},\ldots,x^{(N)}\}$
- Complex model parameterized with θ
- There is a latent space z with

$$z \sim p(z) \quad \text{multivariate Gaussian}$$

$$x|z \sim p_\theta(x|z)$$

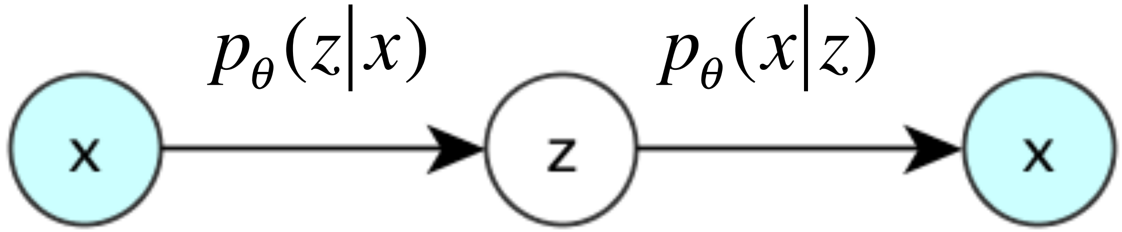


$$z \xrightarrow{p_\theta(X|Z)} x$$

One Example

Wish to learn θ from the N training observations $x^{(i)}$ i=1,…,N

TRAINING AS AN AUTOENCODER



Training use maximum likelihood
of p(x) given the training data

Problem:

$$p_\theta(z|x)$$

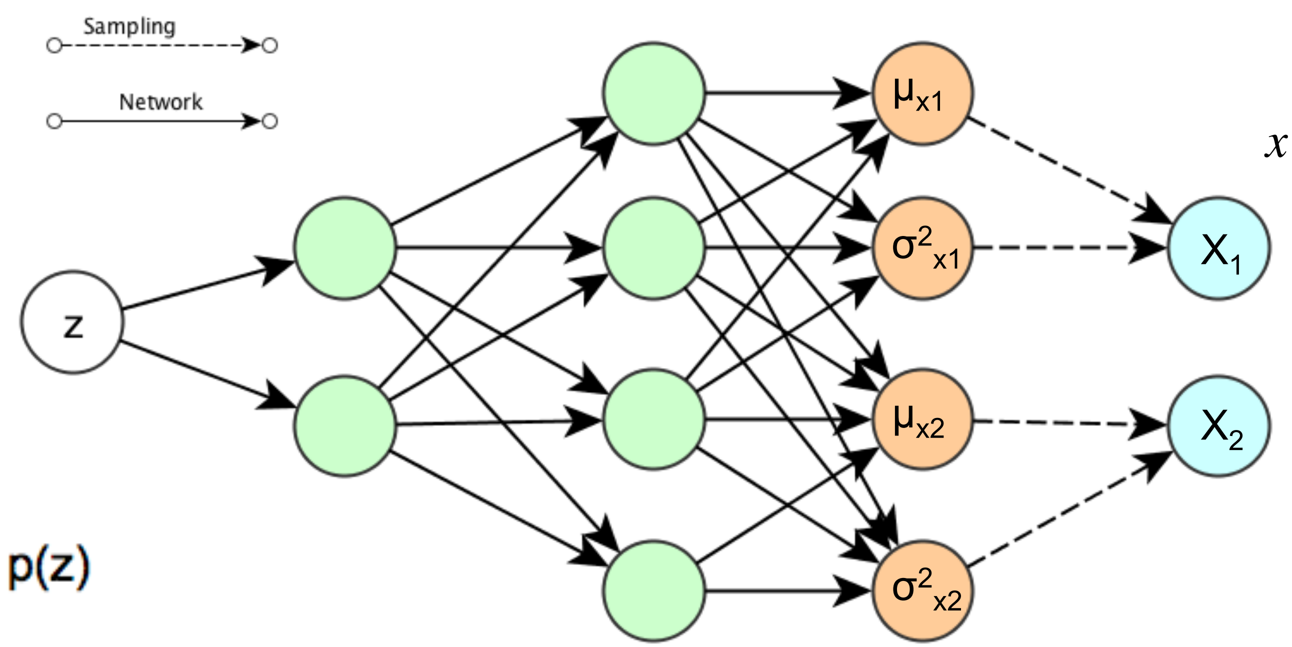
Cannot be calculated:

Solution:

- MCMC (too costly)
- Approximate p(z|x) with q(z|x)

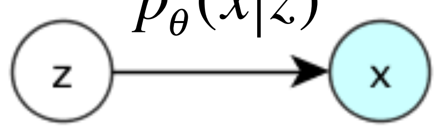
MODEL FOR DECODER NETWORK

- For illustration z one dimensional x 2D
- Want a complex model of distribution of x given z
- Idea: **NN** + Gaussian (or Bernoulli) here with diagonal covariance Σ



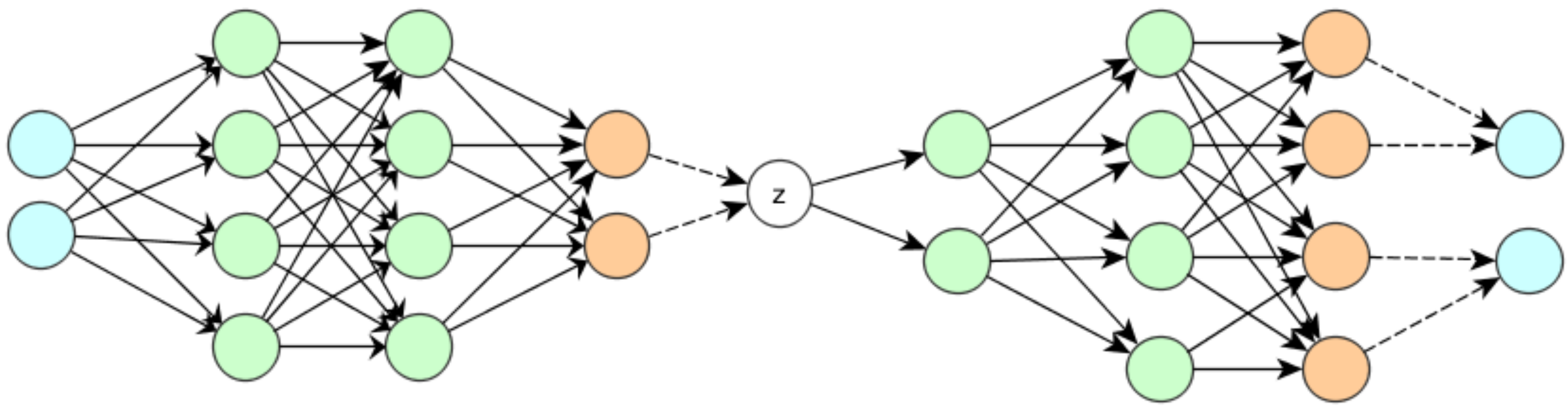
$$x|z \sim N(\mu_x, \sigma_x^{\;2})$$

$$p_\theta(x|z)$$

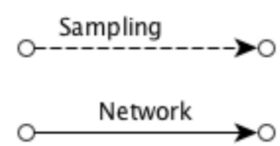


COMPLETE AUTO-ENCODER

$q_\varphi(x|z)$ $p_\theta(x|z)$



Learning the parameters φ and θ via backpropagation



Determining the loss function

TRAINING: LOSS FUNCTION

- What is (one of the) most beautiful idea in statistics?

- Max-Likelihood, tune Φ, θ to maximize the likelihood

- We maximize the (log) likelihood of a given “image” $x^{(i)}$ of the training set. Later we sum over all training data (using minibatches)

LOWER BOUND OF LIKELIHOOD

Likelihood, for an image $x^{(i)}$ from training set. Writing $x=x^{(i)}$ for short.

$$
\begin{aligned}
L &= \log(p(x)) \\
&= \sum_z q(z|x)\ \log(p(x)) \qquad\qquad \text{multiplied with 1} \\
&= \sum_z q(z|x)\ \log\left(\frac{p(z,x)}{p(z|x)}\right) \\
&= \sum_z q(z|x)\ \log\left(\frac{p(z,x)}{q(z|x)}\frac{q(z|x)}{p(z|x)}\right) \\
&= \sum_z q(z|x)\ \log\left(\frac{p(z,x)}{q(z|x)}\right) + \sum_z q(z|x)\ \log\left(\frac{q(z|x)}{p(z|x)}\right) \\
&= L^{\mathrm{v}} + D_{\mathrm{KL}}\left(q(z|x)||p(z|x)\right) \\
&\geq L^{\mathrm{v}}
\end{aligned}
$$

D_{KL} KL-Divergence >= 0 depends on how good q(z|x) can approximate p(z|x)

L^v *"lower variational bound of the (log) likelihood"* L^v =L for perfect approximation

APPROXIMATE INFERENCE

$$
\begin{aligned}
L^{v} &= \sum_{z} q(z|x)\ \log\left(\frac{p(z,x)}{q(z|x)}\right) \qquad \text{with } p(z,x) = p(x|z)\,p(z) \\
&= \sum_{z} q(z|x)\ \log\left(\frac{p(x|z)p(z)}{q(z|x)}\right) \\
&= \sum_{z} q(z|x)\ \log\left(\frac{p(z)}{q(z|x)}\right) + \sum_{z} q(z|x)\ \log\left(p(x|z)\right) \\
&= -D_{\mathrm{KL}}\left(q(z|x)||p(z)\right) + \mathbb{E}_{q(z|x)}\left(\log\left(p(x|z)\right)\right) \qquad \text{putting in } x^{(i)} \text{ for } x \\
&= -D_{\mathrm{KL}}\left(q(z|x^{(i)})||p(z)\right) + \mathbb{E}_{q(z|x^{(i)})}\left(\log\left(p(x^{(i)}|z)\right)\right)
\end{aligned}
$$

Regularisation
p(z) is usually a simple prior N(0,1)

Reconstruction quality, log(1) if $x^{(i)}$ gets always reconstructed perfectly (z produces $x^{(i)}$)

Example $x^{(i)}$

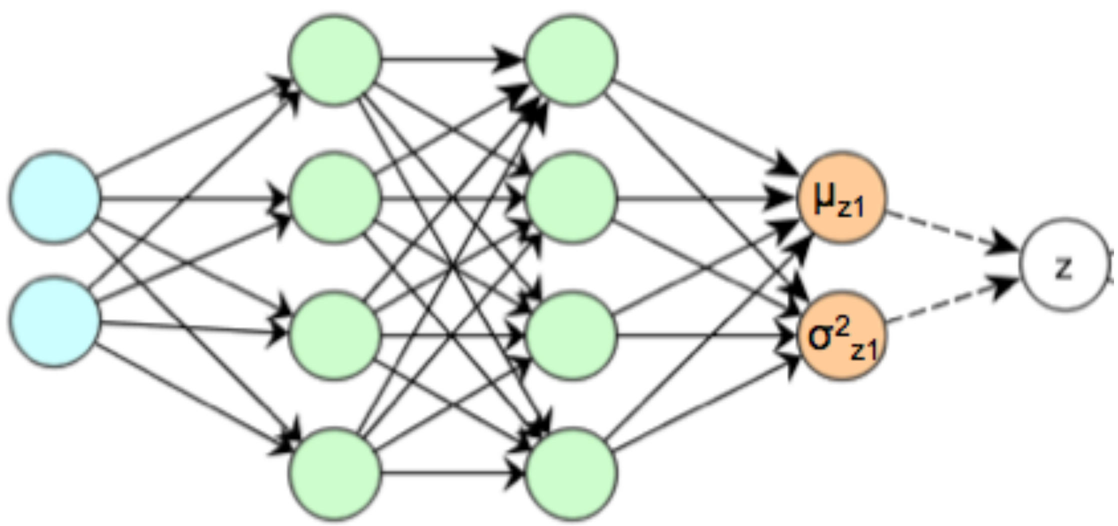
$$q_\phi(z|x^{(i)}) \longrightarrow z \qquad p_\theta(x^{(i)}|z) \longrightarrow$$

CALCULATION OF THE REGULARIZATION $-D_{KL}\left(q(z|x^{(i)})||p(z)\right)$

Use N(0,1) as prior for p(z)

$q(z|x^{(i)})$ is Gaussian with parameters $(\mu^{(i)},\sigma^{(i)})$ determined by NN

$$-D_{KL}\left(q(z|x^{(i)})||p(z)\right) = \frac{1}{2}\sum_{j=1}^{J}\left(1 + \log(\sigma_{z_j}^{(i)^2}) - \mu_{z_j}^{(i)^2} - \sigma_{z_j}^{(i)^2}\right)$$



SAMPLING TO CALCULATE $\mathbb{E}_{q(z|x^{(i)})}\left(\log\left(p(x^{(i)}|z)\right)\right)$

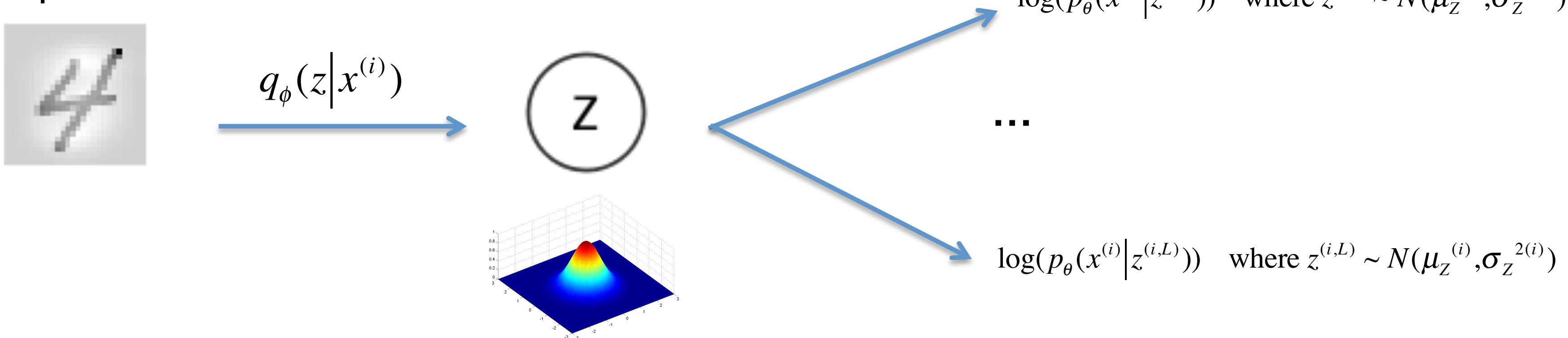
Approximating $\mathbb{E}_{q(z|x^{(i)})}$ with sampling form the distribution $q(z|x^{(i)})$

With $z^{(i,l)}$ $l = 1, 2, \ldots L$ sampled from $z^{(i,l)} \sim q(z|x^{(i)})$

$$L^{\mathrm{v}} = -D_{\mathrm{KL}}\left(q(z|x^{(i)})||p(z)\right) + \mathbb{E}_{q(z|x^{(i)})}\left(\log\left(p(x^{(i)}|z)\right)\right)$$

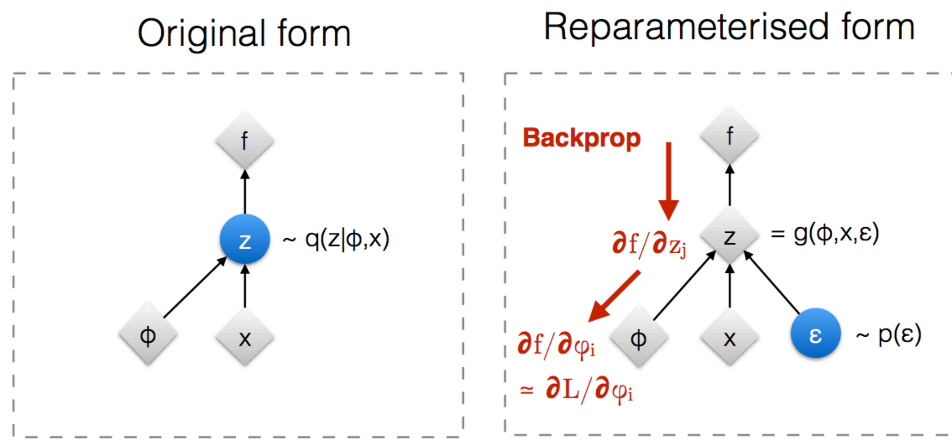
$$L^{\mathrm{v}} \approx -D_{\mathrm{KL}}\left(q(z|x^{(i)})||p(z)\right) + \frac{1}{L}\sum_{i=1}^{L}\log\left(p(x^{(i)}|z^{(i,l)})\right)$$

Example $x^{(i)}$



AN USEFUL TRICK

Backpropagation not possible through random sampling!



: Deterministic node

: Random node

[Kingma, 2013]
[Bengio, 2013]
[Kingma and Welling 2014]
[Rezende et al 2014]

Sampling (reparametrization trick)

Cannot back propagate through a random drawn number

$$z^{(i,l)} \sim N(\mu^{(i)}, \sigma^{2(i)})$$

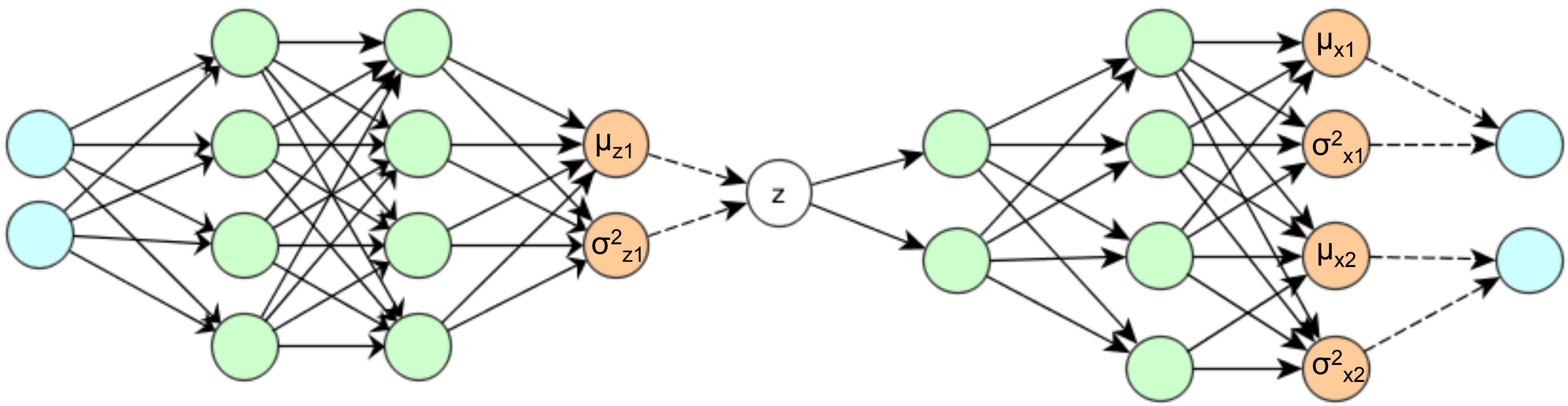
$$z^{(i,l)} = \mu^{(i)} + \sigma^{(i)} \odot \varepsilon_i \quad \varepsilon_i \sim N(0,1)$$

z has the same distribution, but now one can back propagate.

Writing z in this form, results in a deterministic part and noise.

PUTTING IT ALL TOGETHER

Prior p(z) ~ N(0,1) and p, q Gaussian, extension to dim(z) > 1 trivial



Cost: Regularisation

$$-D_{KL}\left(q(z|x^{(i)})||p(z)\right) = \frac{1}{2}\sum_{j=1}^{J}\left(1 + \log(\sigma_{z_j}^{(i)^2}) - \mu_{z_j}^{(i)^2} - \sigma_{z_j}^{(i)^2}\right)$$

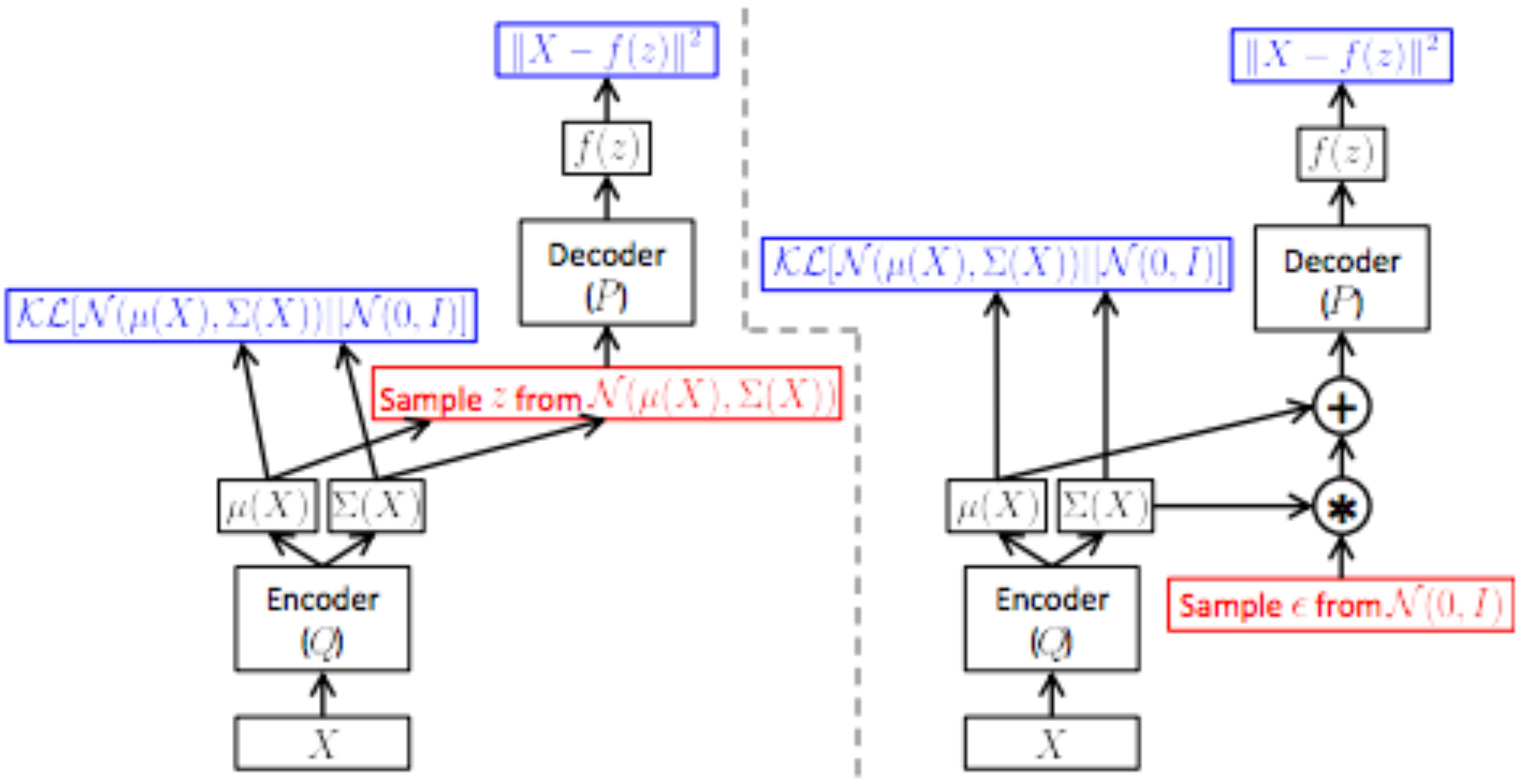
Cost: Reproduction

$$-\log\left(p(x^{(i)}|z^{(i)})\right) = \sum_{j=1}^{D}\frac{1}{2}\log(\sigma_{x_j}^2) + \frac{(x_j^{(i)} - \mu_{x_j})^2}{2\sigma_{x_j}^2}$$

We use mini batch gradient decent to optimize the cost function over all $x^{(i)}$ in the mini batch

Least Square for constant variance

PUTTING IT ALL TOGETHER



Lecture 4

Denoising Auto-encoders

INTRODUCTION

Denoising Autoencoders for learning Deep Networks

For more details, see:

P. Vincent, H. Larochelle, Y. Bengio, P.A. Manzagol, **Extracting and Composing Robust Features with Denoising Autoencoders**, *Proceedings of the 25th International Conference on Machine Learning (ICML'2008)*, pp. 1096-1103, Omnipress, 2008.

INTRODUCTION

- Building good predictors on complex domains means learning complicated functions.
- These are best represented by multiple levels of non-linear operations i.e. deep architectures.
- Deep architectures are an old idea: multi-layer perceptrons.
- Learning the parameters of deep architectures proved to be challenging!

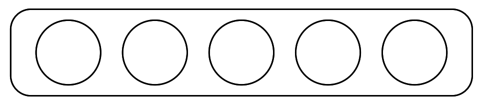
MAIN IDEA

Open question: what would make a good unsupervised criterion for finding good initial intermediate representations?

- Inspiration: our ability to "fill-in-the-blanks" in sensory input.
  missing pixels, small occlusions, image from sound, ...
- Good fill-in-the-blanks performance $\leftrightarrow$ distribution is well captured.
- $\longrightarrow$ old notion of associative memory (motivated Hopfield models (Hopfield, 1982))

What we propose:
unsupervised initialization by explicit fill-in-the-blanks training.

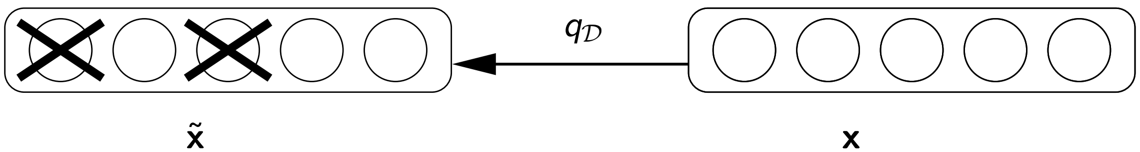
DENOISING AUTOENCODER



$\mathbf{x}$

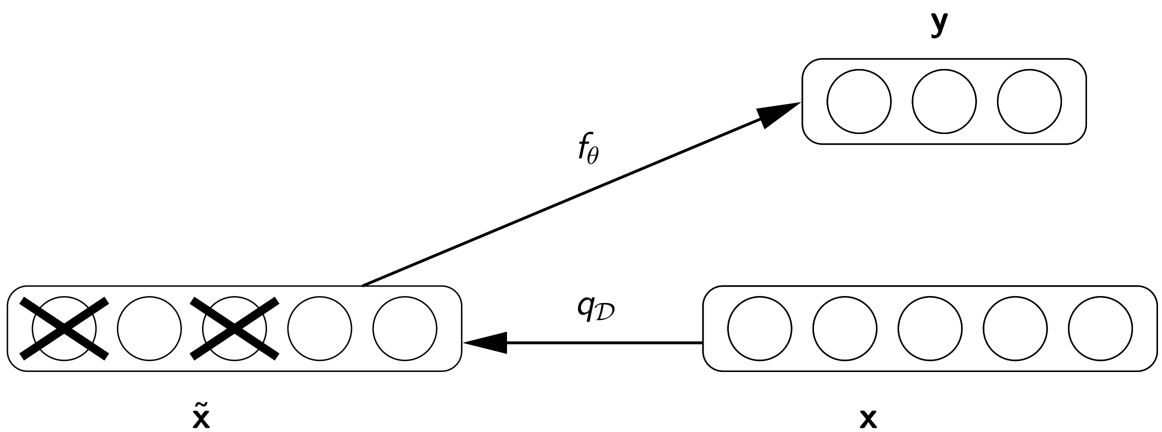
- Clean input $\mathbf{x} \in [0,1]^d$ is partially destroyed, yielding corrupted input: $\tilde{\mathbf{x}} \sim q_{\mathcal{D}}(\tilde{\mathbf{x}}|\mathbf{x})$.
- $\tilde{\mathbf{x}}$ is mapped to hidden representation $\mathbf{y} = f_\theta(\tilde{\mathbf{x}})$.
- From $\mathbf{y}$ we reconstruct a $\mathbf{z} = g_{\theta'}(\mathbf{y})$.
- Train parameters to minimize the cross-entropy "reconstruction error" $L_{\mathbb{H}}(\mathbf{x},\mathbf{z}) = \mathbb{H}(\mathcal{B}_{\mathbf{x}} \| \mathcal{B}_{\mathbf{z}})$, where $\mathcal{B}_{\mathbf{x}}$ denotes multivariate Bernoulli distribution with parameter $\mathbf{x}$.

DENOISING AUTOENCODER



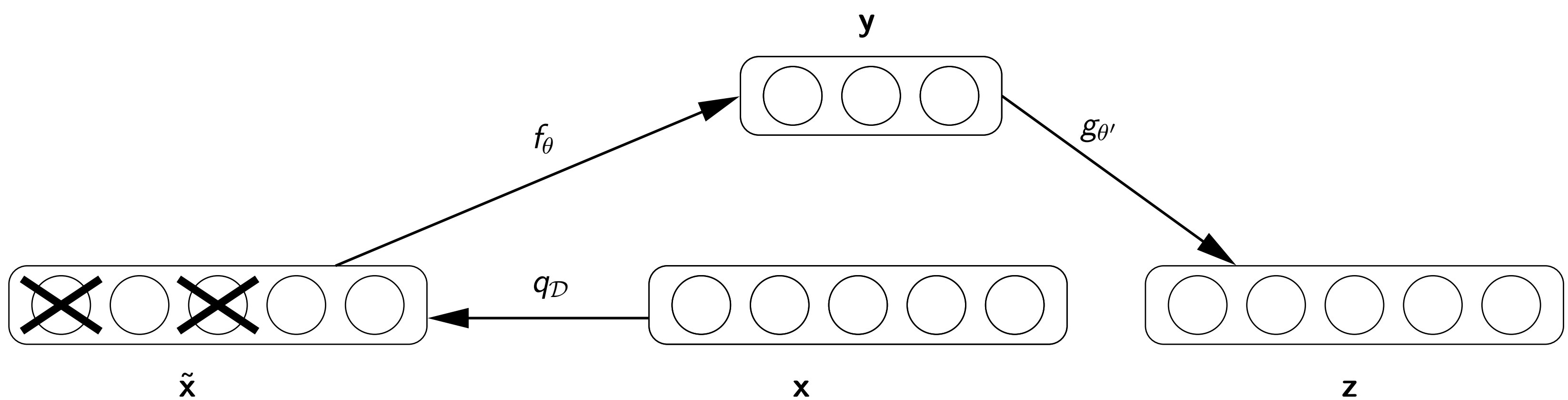
- Clean input $\mathbf{x} \in [0,1]^d$ is partially destroyed, yielding corrupted input: $\tilde{\mathbf{x}} \sim q_{\mathcal{D}}(\tilde{\mathbf{x}}|\mathbf{x})$.
- $\tilde{\mathbf{x}}$ is mapped to hidden representation $\mathbf{y} = f_\theta(\tilde{\mathbf{x}})$.
- From $\mathbf{y}$ we reconstruct a $\mathbf{z} = g_{\theta'}(\mathbf{y})$.
- Train parameters to minimize the cross-entropy "reconstruction error" $L_{\mathbb{H}}(\mathbf{x}, \mathbf{z}) = \mathbb{H}(\mathcal{B}_{\mathbf{x}} \| \mathcal{B}_{\mathbf{z}})$, where $\mathcal{B}_{\mathbf{x}}$ denotes multivariate Bernoulli distribution with parameter $\mathbf{x}$.

DENOISING AUTOENCODER



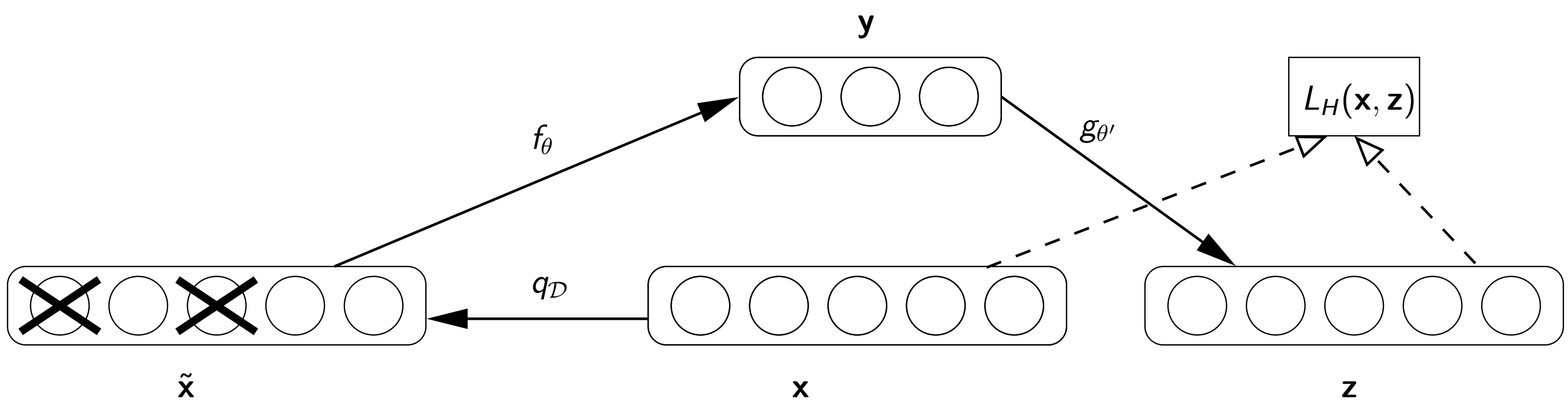
- Clean input $\mathbf{x} \in [0,1]^d$ is partially destroyed, yielding corrupted input: $\tilde{\mathbf{x}} \sim q_\mathcal{D}(\tilde{\mathbf{x}}|\mathbf{x})$.
- $\tilde{\mathbf{x}}$ is mapped to hidden representation $\mathbf{y} = f_\theta(\tilde{\mathbf{x}})$.
- From $\mathbf{y}$ we reconstruct a $\mathbf{z} = g_{\theta'}(\mathbf{y})$.
- Train parameters to minimize the cross-entropy "reconstruction error" $L_{\mathbb{H}}(\mathbf{x},\mathbf{z}) = \mathbb{H}(\mathcal{B}_\mathbf{x} \| \mathcal{B}_\mathbf{z})$, where $\mathcal{B}_\mathbf{x}$ denotes multivariate Bernoulli distribution with parameter $\mathbf{x}$.

DENOISING AUTOENCODER



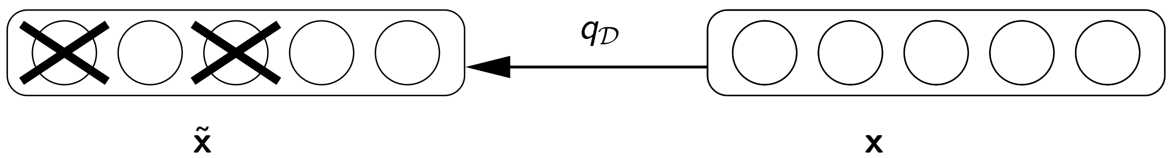
- Clean input $\mathbf{x} \in [0,1]^d$ is partially destroyed, yielding corrupted input: $\tilde{\mathbf{x}} \sim q_\mathcal{D}(\tilde{\mathbf{x}}|\mathbf{x})$.
- $\tilde{\mathbf{x}}$ is mapped to hidden representation $\mathbf{y} = f_\theta(\tilde{\mathbf{x}})$.
- From $\mathbf{y}$ we reconstruct a $\mathbf{z} = g_{\theta'}(\mathbf{y})$.
- Train parameters to minimize the cross-entropy "reconstruction error" $L_{\mathbb{H}}(\mathbf{x},\mathbf{z}) = \mathbb{H}(\mathcal{B}_\mathbf{x} \| \mathcal{B}_\mathbf{z})$, where $\mathcal{B}_\mathbf{x}$ denotes multivariate Bernoulli distribution with parameter $\mathbf{x}$.

DENOISING AUTOENCODER



- Clean input $\mathbf{x} \in [0,1]^d$ is partially destroyed, yielding corrupted input: $\tilde{\mathbf{x}} \sim q_\mathcal{D}(\tilde{\mathbf{x}}|\mathbf{x})$.
- $\tilde{\mathbf{x}}$ is mapped to hidden representation $\mathbf{y} = f_\theta(\tilde{\mathbf{x}})$.
- From $\mathbf{y}$ we reconstruct a $\mathbf{z} = g_{\theta'}(\mathbf{y})$.
- Train parameters to minimize the cross-entropy "reconstruction error" $L_{\mathbb{H}}(\mathbf{x},\mathbf{z}) = \mathbb{H}(\mathcal{B}_\mathbf{x} \| \mathcal{B}_\mathbf{z})$, where $\mathcal{B}_\mathbf{x}$ denotes multivariate Bernoulli distribution with parameter $\mathbf{x}$.

NOISE PROCESS



- Choose a fixed proportion ν of components of $\mathbf{x}$ at random.
- Reset their values to 0.
- Can be viewed as replacing a component considered missing by a default value.

Other corruption processes are possible.

ENCODER – DECODER

We use standard sigmoid network layers:

- $\mathbf{y} = f_\theta(\tilde{\mathbf{x}}) = \text{sigmoid}(\underbrace{\mathbf{W}}_{d' \times d} \tilde{\mathbf{x}} + \underbrace{\mathbf{b}}_{d' \times 1})$
- $g_{\theta'}(\mathbf{y}) = \text{sigmoid}(\underbrace{\mathbf{W}'}_{d \times d'} \mathbf{y} + \underbrace{\mathbf{b}'}_{d \times 1})$.

and cross-entropy loss.

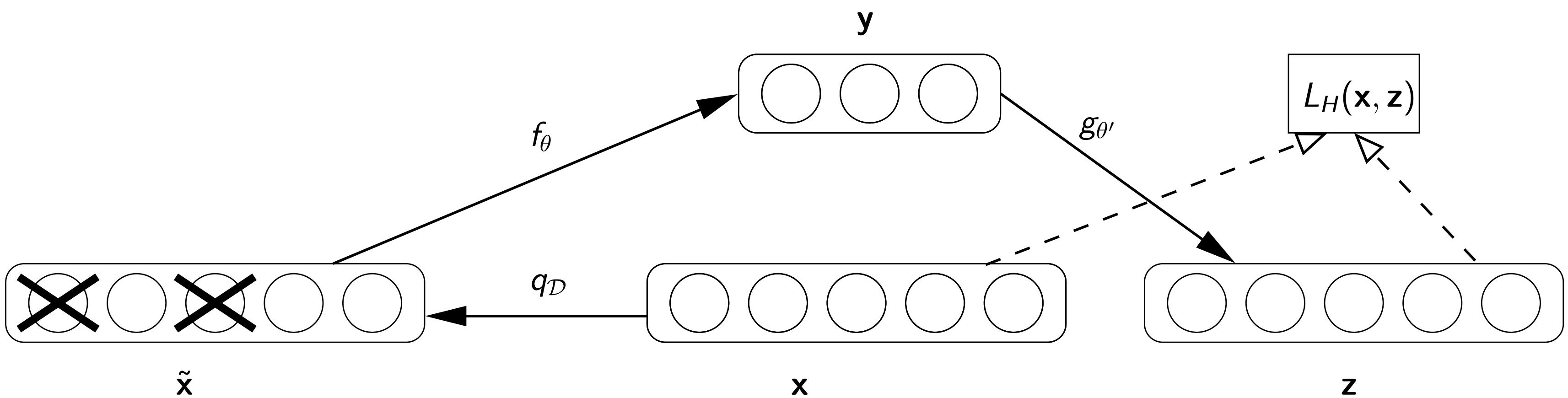
ENCODER – DECODER

Denoising is a fundamentally different task

- Think of classical autoencoder in overcomplete case: $d' \geq d$
- Perfect reconstruction is possible without having learnt anything useful!
- Denoising autoencoder learns useful representation in this case.
- Being good at denoising requires capturing structure in the input.

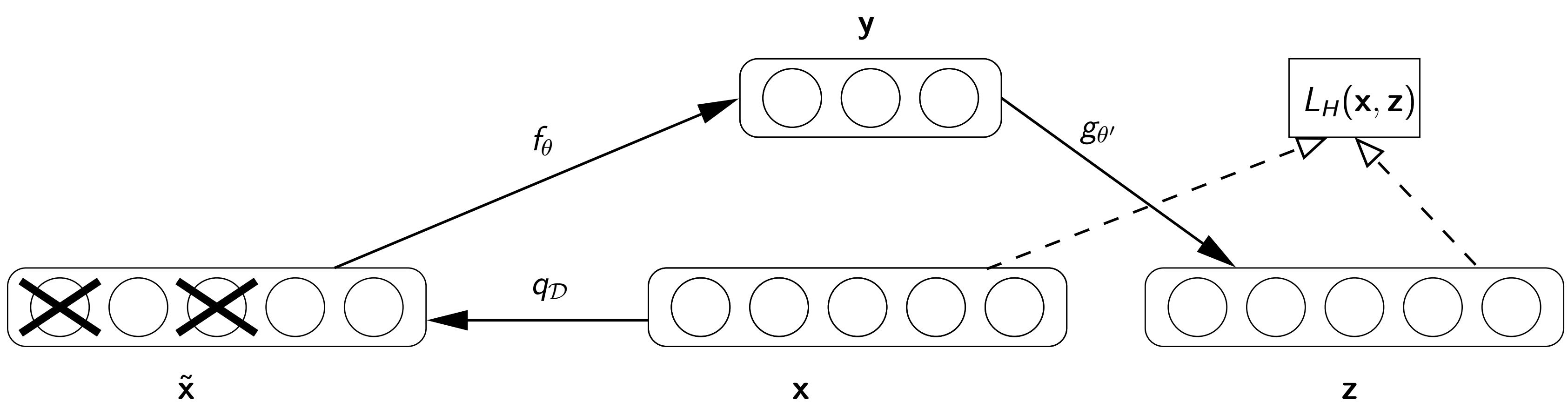
Denoising using classical autoencoders was actually introduced much earlier (LeCun, 1987; Gallinari et al., 1987), as an alternative to Hopfield networks (Hopfield, 1982).

LAYER-WISE INITIALIZATION



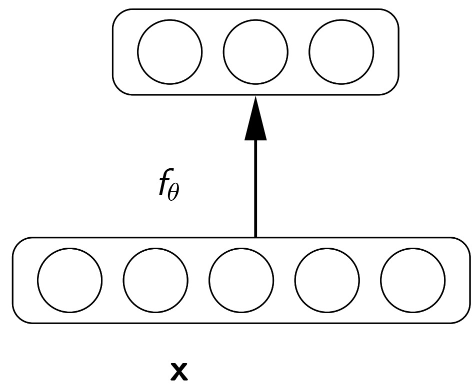
1. Learn first mapping f_θ by training as a denoising autoencoder.
2. Remove scaffolding. Use f_θ directly on input yielding higher level representation.
3. Learn next level mapping $f_\theta^{(2)}$ by training denoising autoencoder on current level representation.
4. Iterate to initialize subsequent layers.

LAYER-WISE INITIALIZATION



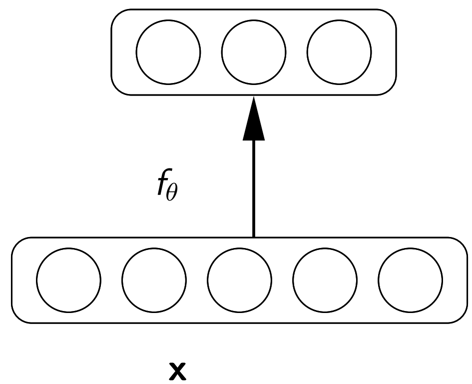
1. Learn first mapping f_θ by training as a denoising autoencoder.
2. Remove scaffolding. Use f_θ directly on input yielding higher level representation.
3. Learn next level mapping $f_\theta^{(2)}$ by training denoising autoencoder on current level representation.
4. Iterate to initialize subsequent layers.

LAYER-WISE INITIALIZATION



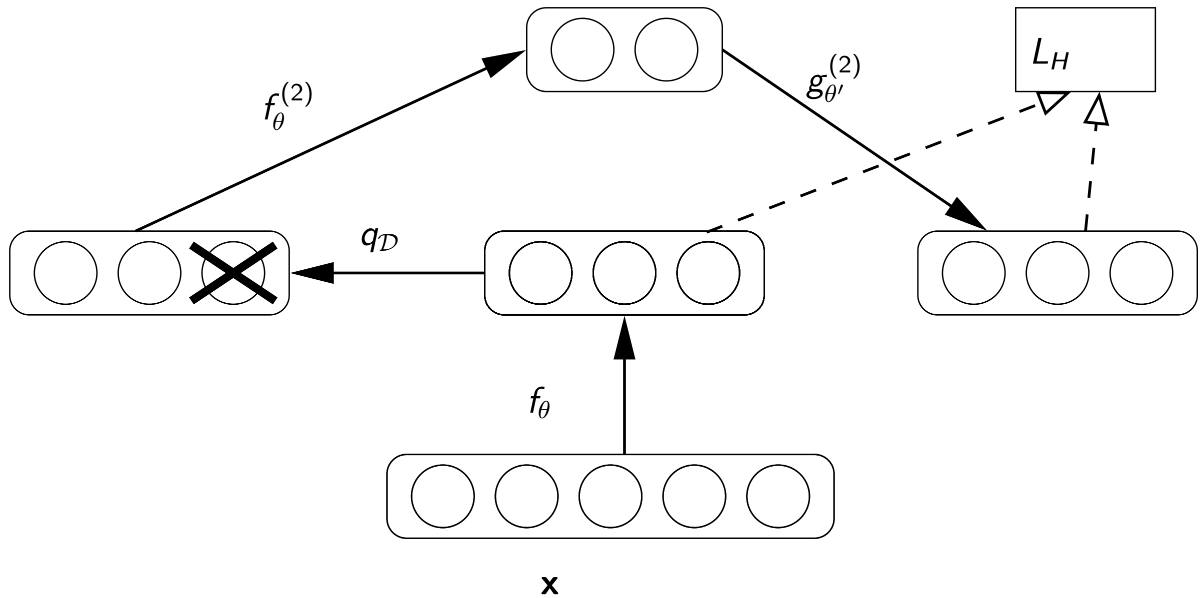
1. Learn first mapping f_θ by training as a denoising autoencoder.
2. Remove scaffolding. Use f_θ directly on input yielding higher level representation.
3. Learn next level mapping $f_\theta^{(2)}$ by training denoising autoencoder on current level representation.
4. Iterate to initialize subsequent layers.

LAYER-WISE INITIALIZATION



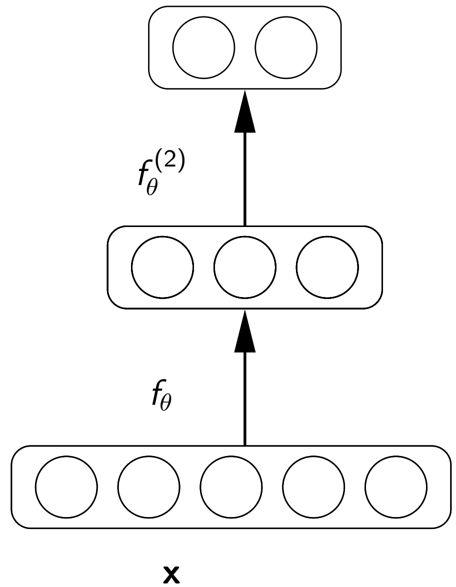
1. Learn first mapping f_θ by training as a denoising autoencoder.
2. Remove scaffolding. Use f_θ directly on input yielding higher level representation.
3. Learn next level mapping $f_\theta^{(2)}$ by training denoising autoencoder on current level representation.
4. Iterate to initialize subsequent layers.

LAYER-WISE INITIALIZATION



1. Learn first mapping f_θ by training as a denoising autoencoder.
2. Remove scaffolding. Use f_θ directly on input yielding higher level representation.
3. Learn next level mapping $f_\theta^{(2)}$ by training denoising autoencoder on current level representation.
4. Iterate to initialize subsequent layers.

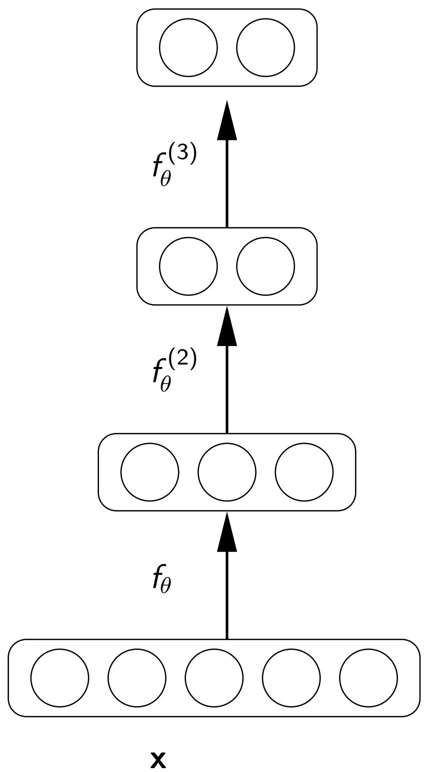
LAYER-WISE INITIALIZATION



1. Learn first mapping f_θ by training as a denoising autoencoder.
2. Remove scaffolding. Use f_θ directly on input yielding higher level representation.
3. Learn next level mapping $f_\theta^{(2)}$ by training denoising autoencoder on current level representation.
4. Iterate to initialize subsequent layers.

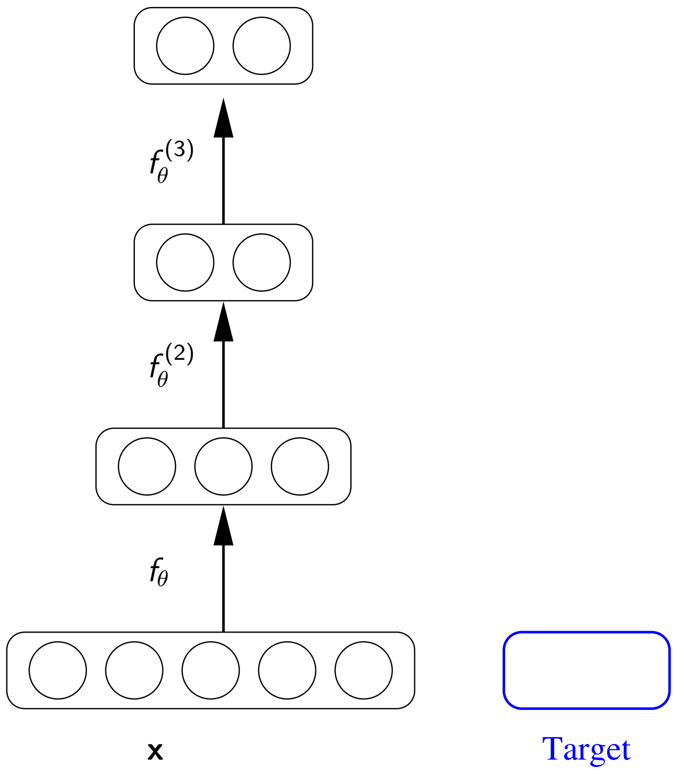
SUPERVISED FINE-TUNING

- Initial deep mapping was learnt in an unsupervised way.
- → initialization for a supervised task.
- Output layer gets added.
- Global fine tuning by gradient descent on supervised criterion.



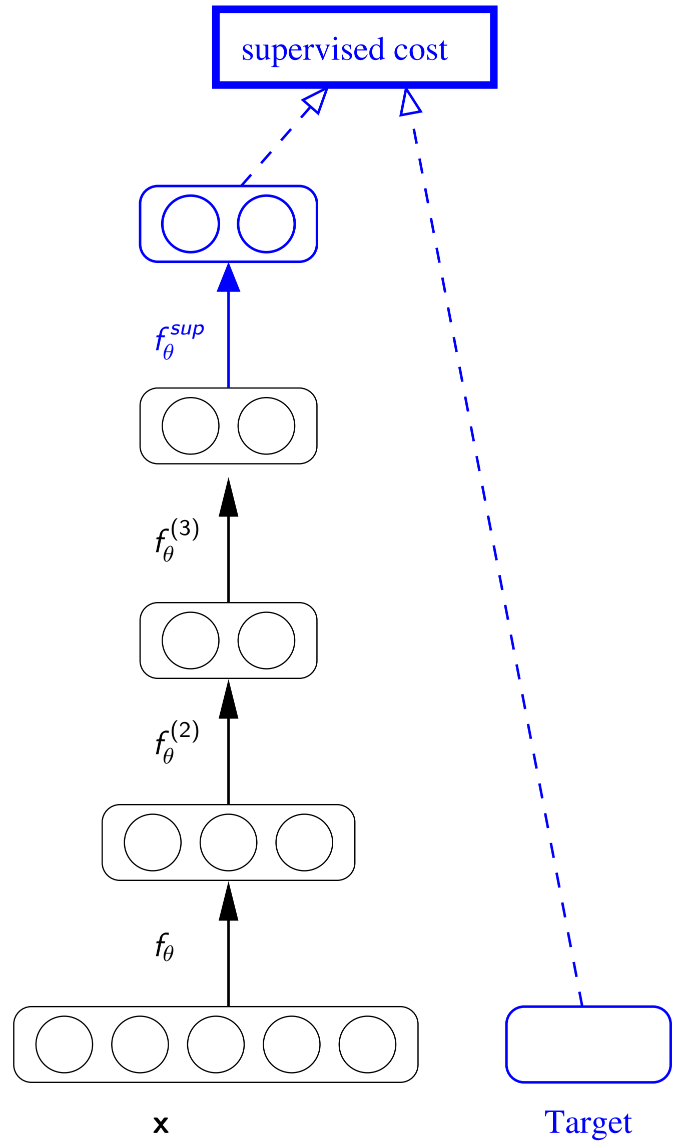
SUPERVISED FINE-TUNING

- Initial deep mapping was learnt in an unsupervised way.
- → initialization for a supervised task.
- Output layer gets added.
- Global fine tuning by gradient descent on supervised criterion.

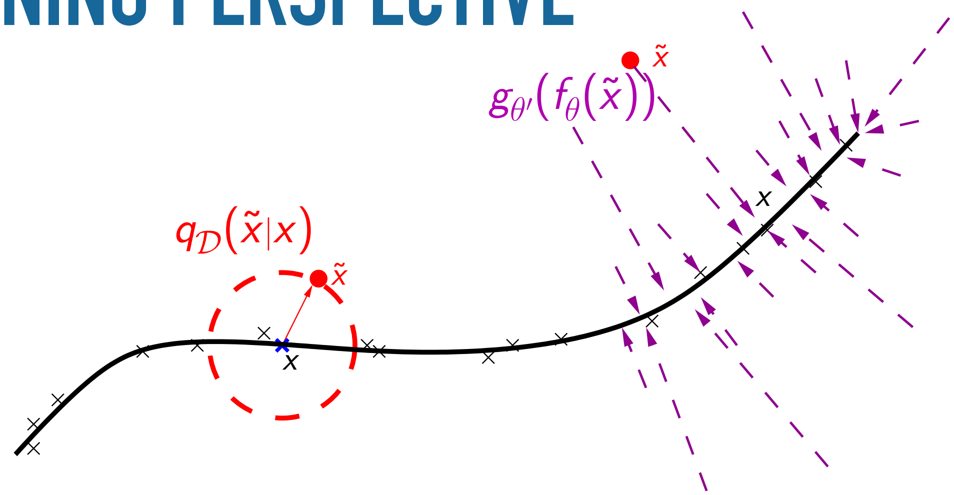


SUPERVISED FINE-TUNING

- Initial deep mapping was learnt in an unsupervised way.
- $\longrightarrow$ initialization for a supervised task.
- Output layer gets added.
- Global fine tuning by gradient descent on supervised criterion.



MANIFOLD LEARNING PERSPECTIVE



Denoising autoencoder can be seen as a way to learn a manifold:

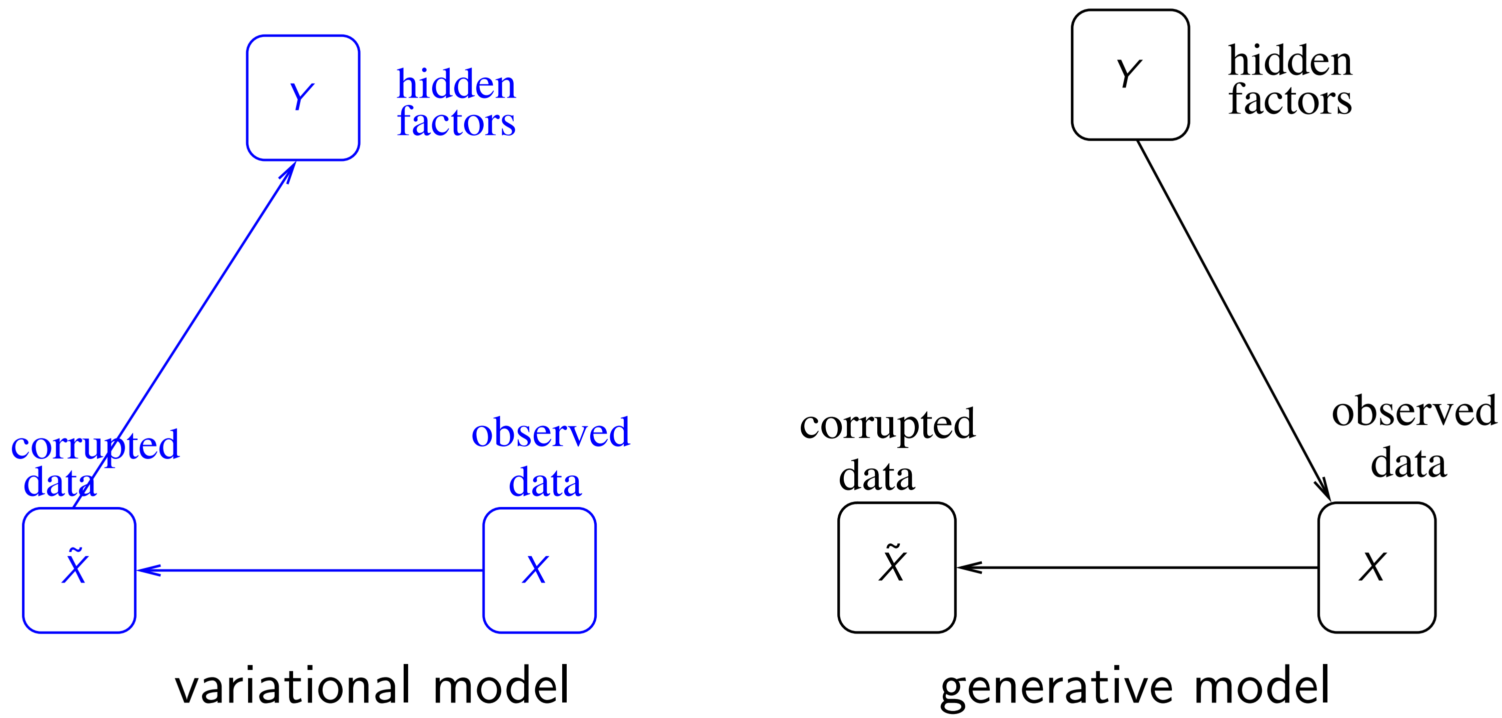
- Suppose training data ($\times$) concentrate near a low-dimensional manifold.
- Corrupted examples (•) are obtained by applying corruption process $q_{\mathcal{D}}(\widetilde{X}|X)$ and will lie farther from the manifold.
- The model learns with $p(X|\widetilde{X})$ to "project them back" onto the manifold.
- Intermediate representation Y can be interpreted as a coordinate system for points on the manifold.

INFORMATION THEORETIC PERSPECTIVE

- Consider $X \sim q(X)$, q unknown. $\widetilde{X} \sim q_{\mathcal{D}}(\widetilde{X}|X)$. $Y = f_\theta(\widetilde{X})$.
- It can be shown that minimizing the expected reconstruction error amounts to maximizing a lower bound on mutual information $\mathbb{I}(X; Y)$.
- Denoising autoencoder training can thus be justified by the objective that hidden representation Y captures as much information as possible about X *even as Y is a function of corrupted input.*

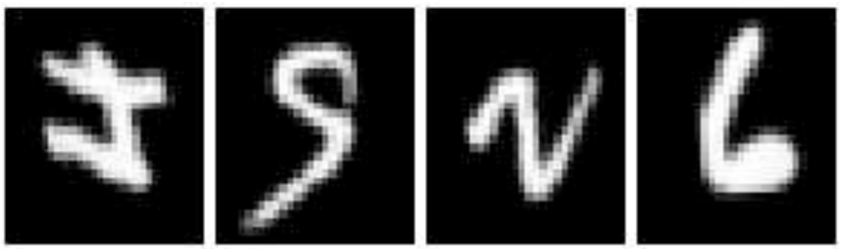
GENERATIVE MODEL PERSPECTIVE

- Denoising autoencoder training can be shown to be equivalent to maximizing a variational bound on the likelihood of a generative model for the corrupted data.



VARIATIONS ON MNIST DIGIT CLASSIFICATION

basic: subset of original MNIST digits: 10 000 training samples, 2 000 validation samples, 50 000 test samples.



rot: applied random rotation (angle between 0 and 2π radians)

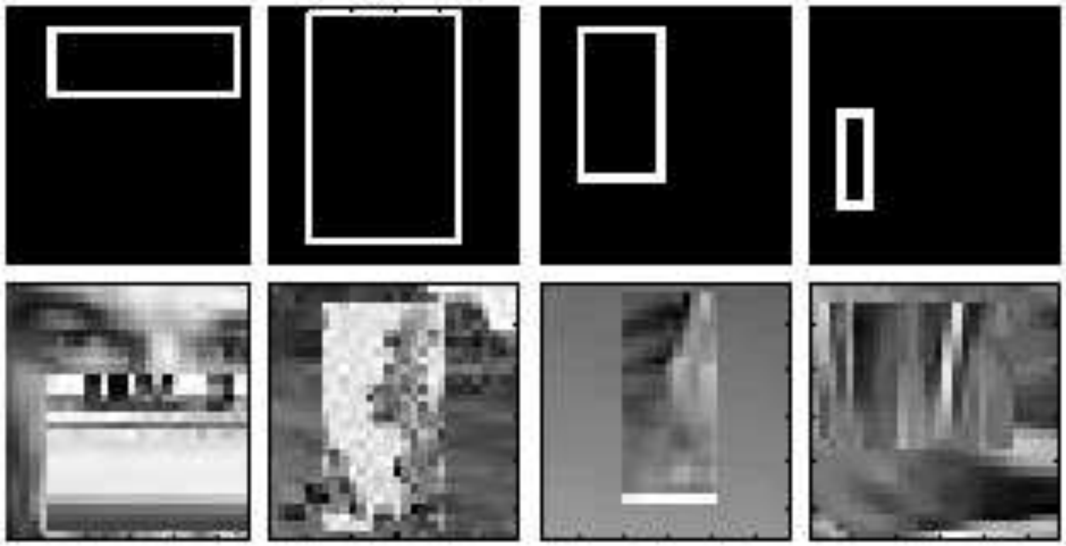
bg-rand: background made of random pixels (value in $0 \ldots 255$)

bg-img: background is random patch from one of 20 images

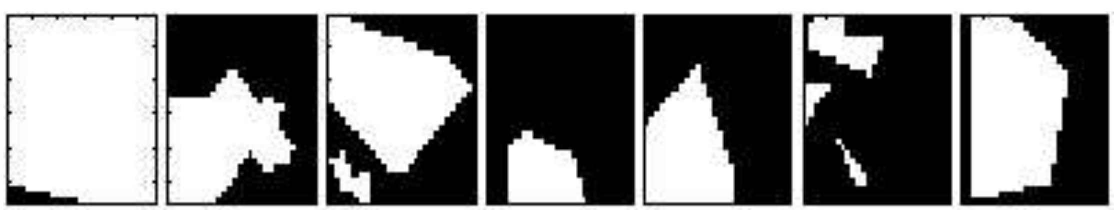
rot-bg-img: combination of rotation and background image

SHAPE DISCRIMINATION

- **rect:** discriminate between tall and wide rectangles on black background.



- **rect-img**: borderless rectangle filled with random image patch. Background is a different image patch.
- **convex:** discriminate between convex and non-convex shapes.



EXPERIMENTATION

We compared the following algorithms on the benchmark problems:

- **SVM**$_{rbf}$: suport Vector Machines with Gaussian Kernel.
- **DBN-3**: Deep Belief Nets with 3 hidden layers (stacked Restricted Boltzmann Machines trained with contrastive divergence).
- **SAA-3**: Stacked Autoassociators with 3 hidden layers (no denoising).
- **SdA-3**: Stacked Denoising Autoassociators with 3 hidden layers.

Hyper-parameters for all algorithms were tuned based on classificaiton performance on validation set. (In particular hidden-layer sizes, and ν for **SdA-3**).

PERFORMANCE COMPARISON

Dataset	**SVM**$_{rbf}$	**DBN-3**	**SAA-3**	**SdA-3** (ν)	**SVM**$_{rbf}(\nu)$
basic	$3.03_{\pm 0.15}$	$3.11_{\pm 0.15}$	$3.46_{\pm 0.16}$	$2.80_{\pm 0.14}$ (10%)	3.07 (10%)
rot	$11.11_{\pm 0.28}$	$10.30_{\pm 0.27}$	$10.30_{\pm 0.27}$	$10.29_{\pm 0.27}$ (10%)	11.62 (10%)
bg-rand	$14.58_{\pm 0.31}$	$6.73_{\pm 0.22}$	$11.28_{\pm 0.28}$	$10.38_{\pm 0.27}$ (40%)	15.63 (25%)
bg-img	$22.61_{\pm 0.37}$	$16.31_{\pm 0.32}$	$23.00_{\pm 0.37}$	$16.68_{\pm 0.33}$ (25%)	23.15 (25%)
rot-bg-img	$55.18_{\pm 0.44}$	$47.39_{\pm 0.44}$	$51.93_{\pm 0.44}$	$44.49_{\pm 0.44}$ (25%)	54.16 (10%)
rect	$2.15_{\pm 0.13}$	$2.60_{\pm 0.14}$	$2.41_{\pm 0.13}$	$1.99_{\pm 0.12}$ (10%)	2.45 (25%)
rect-img	$24.04_{\pm 0.37}$	$22.50_{\pm 0.37}$	$24.05_{\pm 0.37}$	$21.59_{\pm 0.36}$ (25%)	23.00 (10%)
convex	$19.13_{\pm 0.34}$	$18.63_{\pm 0.34}$	$18.41_{\pm 0.34}$	$19.06_{\pm 0.34}$ (10%)	24.20 (10%)

PERFORMANCE COMPARISON

Dataset	**SVM$_{rbf}$**	**DBN-3**	**SAA-3**	**SdA-3** (ν)	**SVM$_{rbf}$** (ν)
basic	3.03$\pm$0.15	3.11$\pm$0.15	3.46$\pm$0.16	2.80$\pm$0.14 (10%)	3.07 (10%)
rot	11.11$\pm$0.28	10.30$\pm$0.27	10.30$\pm$0.27	10.29$\pm$0.27 (10%)	11.62 (10%)
bg-rand	14.58$\pm$0.31	6.73$\pm$0.22	11.28$\pm$0.28	10.38$\pm$0.27 (40%)	15.63 (25%)
bg-img	22.61$\pm$0.37	16.31$\pm$0.32	23.00$\pm$0.37	16.68$\pm$0.33 (25%)	23.15 (25%)
rot-bg-img	55.18$\pm$0.44	47.39$\pm$0.44	51.93$\pm$0.44	44.49$\pm$0.44 (25%)	54.16 (10%)
rect	2.15$\pm$0.13	2.60$\pm$0.14	2.41$\pm$0.13	1.99$\pm$0.12 (10%)	2.45 (25%)
rect-img	24.04$\pm$0.37	22.50$\pm$0.37	24.05$\pm$0.37	21.59$\pm$0.36 (25%)	23.00 (10%)
convex	19.13$\pm$0.34	18.63$\pm$0.34	18.41$\pm$0.34	19.06$\pm$0.34 (10%)	24.20 (10%)

PERFORMANCE COMPARISON

Dataset	**SVM$_{rbf}$**	**DBN-3**	**SAA-3**	**SdA-3** (ν)	**SVM$_{rbf}$**(ν)
basic	3.03$_{\pm 0.15}$	3.11$_{\pm 0.15}$	3.46$_{\pm 0.16}$	2.80$_{\pm 0.14}$ (10%)	3.07 (10%)
rot	11.11$_{\pm 0.28}$	10.30$_{\pm 0.27}$	10.30$_{\pm 0.27}$	10.29$_{\pm 0.27}$ (10%)	11.62 (10%)
bg-rand	14.58$_{\pm 0.31}$	6.73$_{\pm 0.22}$	11.28$_{\pm 0.28}$	10.38$_{\pm 0.27}$ (40%)	15.63 (25%)
bg-img	22.61$_{\pm 0.37}$	16.31$_{\pm 0.32}$	23.00$_{\pm 0.37}$	16.68$_{\pm 0.33}$ (25%)	23.15 (25%)
rot-bg-img	55.18$_{\pm 0.44}$	47.39$_{\pm 0.44}$	51.93$_{\pm 0.44}$	44.49$_{\pm 0.44}$ (25%)	54.16 (10%)
rect	2.15$_{\pm 0.13}$	2.60$_{\pm 0.14}$	2.41$_{\pm 0.13}$	1.99$_{\pm 0.12}$ (10%)	2.45 (25%)
rect-img	24.04$_{\pm 0.37}$	22.50$_{\pm 0.37}$	24.05$_{\pm 0.37}$	21.59$_{\pm 0.36}$ (25%)	23.00 (10%)
convex	19.13$_{\pm 0.34}$	18.63$_{\pm 0.34}$	18.41$_{\pm 0.34}$	19.06$_{\pm 0.34}$ (10%)	24.20 (10%)

PERFORMANCE COMPARISON

Dataset	SVM$_{rbf}$	DBN-3	SAA-3	SdA-3 (ν)	SVM$_{rbf}$(ν)
basic	3.03$_{\pm 0.15}$	3.11$_{\pm 0.15}$	3.46$_{\pm 0.16}$	2.80$_{\pm 0.14}$ (10%)	3.07 (10%)
rot	11.11$_{\pm 0.28}$	10.30$_{\pm 0.27}$	10.30$_{\pm 0.27}$	10.29$_{\pm 0.27}$ (10%)	11.62 (10%)
bg-rand	14.58$_{\pm 0.31}$	6.73$_{\pm 0.22}$	11.28$_{\pm 0.28}$	10.38$_{\pm 0.27}$ (40%)	15.63 (25%)
bg-img	22.61$_{\pm 0.37}$	16.31$_{\pm 0.32}$	23.00$_{\pm 0.37}$	16.68$_{\pm 0.33}$ (25%)	23.15 (25%)
rot-bg-img	55.18$_{\pm 0.44}$	47.39$_{\pm 0.44}$	51.93$_{\pm 0.44}$	44.49$_{\pm 0.44}$ (25%)	54.16 (10%)
rect	2.15$_{\pm 0.13}$	2.60$_{\pm 0.14}$	2.41$_{\pm 0.13}$	1.99$_{\pm 0.12}$ (10%)	2.45 (25%)
rect-img	24.04$_{\pm 0.37}$	22.50$_{\pm 0.37}$	24.05$_{\pm 0.37}$	21.59$_{\pm 0.36}$ (25%)	23.00 (10%)
convex	19.13$_{\pm 0.34}$	18.63$_{\pm 0.34}$	18.41$_{\pm 0.34}$	19.06$_{\pm 0.34}$ (10%)	24.20 (10%)

PERFORMANCE COMPARISON

Dataset	SVM_{rbf}	DBN-3	SAA-3	SdA-3 (ν)	$\text{SVM}_{rbf}(\nu)$
basic	3.03$\pm$0.15	3.11$\pm$0.15	3.46$\pm$0.16	2.80$\pm$0.14 (10%)	3.07 (10%)
rot	11.11$\pm$0.28	10.30$\pm$0.27	10.30$\pm$0.27	10.29$\pm$0.27 (10%)	11.62 (10%)
bg-rand	14.58$\pm$0.31	6.73$\pm$0.22	11.28$\pm$0.28	10.38$\pm$0.27 (40%)	15.63 (25%)
bg-img	22.61$\pm$0.37	16.31$\pm$0.32	23.00$\pm$0.37	16.68$\pm$0.33 (25%)	23.15 (25%)
rot-bg-img	55.18$\pm$0.44	47.39$\pm$0.44	51.93$\pm$0.44	44.49$\pm$0.44 (25%)	54.16 (10%)
rect	2.15$\pm$0.13	2.60$\pm$0.14	2.41$\pm$0.13	1.99$\pm$0.12 (10%)	2.45 (25%)
rect-img	24.04$\pm$0.37	22.50$\pm$0.37	24.05$\pm$0.37	21.59$\pm$0.36 (25%)	23.00 (10%)
convex	19.13$\pm$0.34	18.63$\pm$0.34	18.41$\pm$0.34	19.06$\pm$0.34 (10%)	24.20 (10%)

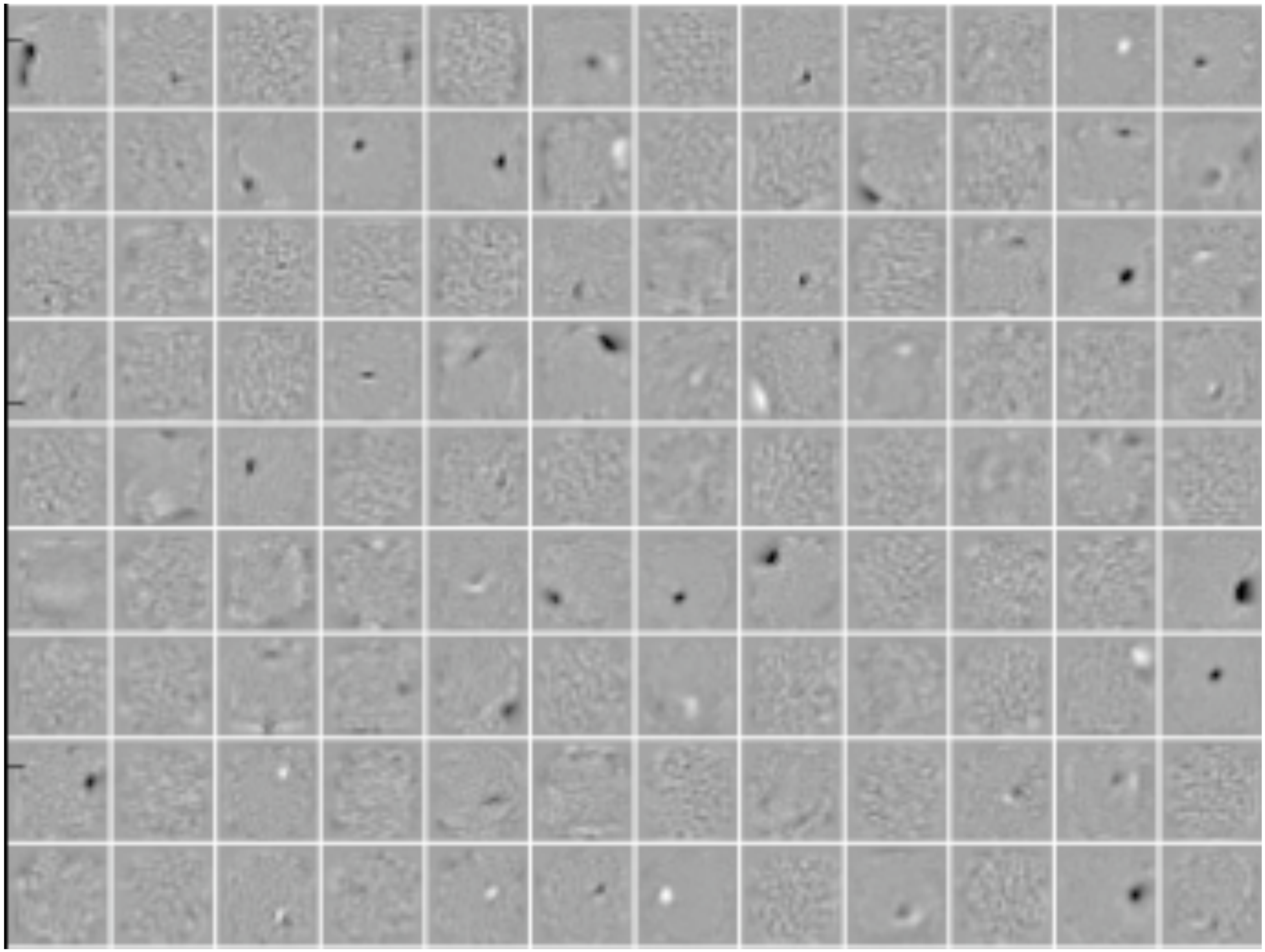
PERFORMANCE COMPARISON

Dataset	SVM$_{rbf}$	DBN-3	SAA-3	SdA-3 (ν)	SVM$_{rbf}$(ν)
basic	3.03$_{\pm 0.15}$	3.11$_{\pm 0.15}$	3.46$_{\pm 0.16}$	2.80$_{\pm 0.14}$ (10%)	3.07 (10%)
rot	11.11$_{\pm 0.28}$	10.30$_{\pm 0.27}$	10.30$_{\pm 0.27}$	10.29$_{\pm 0.27}$ (10%)	11.62 (10%)
bg-rand	14.58$_{\pm 0.31}$	6.73$_{\pm 0.22}$	11.28$_{\pm 0.28}$	10.38$_{\pm 0.27}$ (40%)	15.63 (25%)
bg-img	22.61$_{\pm 0.37}$	16.31$_{\pm 0.32}$	23.00$_{\pm 0.37}$	16.68$_{\pm 0.33}$ (25%)	23.15 (25%)
rot-bg-img	55.18$_{\pm 0.44}$	47.39$_{\pm 0.44}$	51.93$_{\pm 0.44}$	44.49$_{\pm 0.44}$ (25%)	54.16 (10%)
rect	2.15$_{\pm 0.13}$	2.60$_{\pm 0.14}$	2.41$_{\pm 0.13}$	1.99$_{\pm 0.12}$ (10%)	2.45 (25%)
rect-img	24.04$_{\pm 0.37}$	22.50$_{\pm 0.37}$	24.05$_{\pm 0.37}$	21.59$_{\pm 0.36}$ (25%)	23.00 (10%)
convex	19.13$_{\pm 0.34}$	18.63$_{\pm 0.34}$	18.41$_{\pm 0.34}$	19.06$_{\pm 0.34}$ (10%)	24.20 (10%)

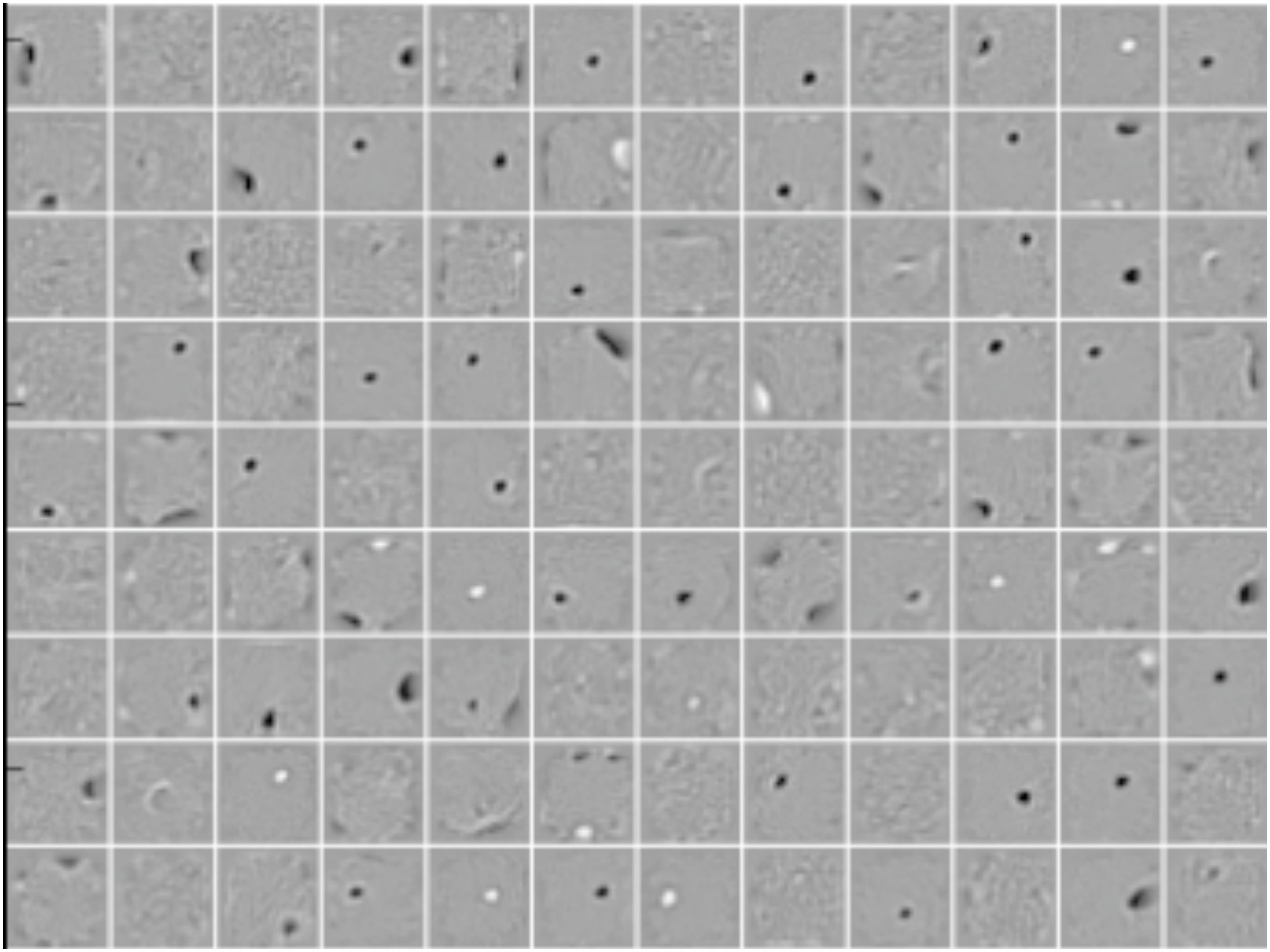
PERFORMANCE COMPARISON

Dataset	SVM$_{rbf}$	DBN-3	SAA-3	SdA-3 (ν)	SVM$_{rbf}$(ν)
basic	3.03$_{\pm 0.15}$	3.11$_{\pm 0.15}$	3.46$_{\pm 0.16}$	2.80$_{\pm 0.14}$ (10%)	3.07 (10%)
rot	11.11$_{\pm 0.28}$	10.30$_{\pm 0.27}$	10.30$_{\pm 0.27}$	10.29$_{\pm 0.27}$ (10%)	11.62 (10%)
bg-rand	14.58$_{\pm 0.31}$	6.73$_{\pm 0.22}$	11.28$_{\pm 0.28}$	10.38$_{\pm 0.27}$ (40%)	15.63 (25%)
bg-img	22.61$_{\pm 0.37}$	16.31$_{\pm 0.32}$	23.00$_{\pm 0.37}$	16.68$_{\pm 0.33}$ (25%)	23.15 (25%)
rot-bg-img	55.18$_{\pm 0.44}$	47.39$_{\pm 0.44}$	51.93$_{\pm 0.44}$	44.49$_{\pm 0.44}$ (25%)	54.16 (10%)
rect	2.15$_{\pm 0.13}$	2.60$_{\pm 0.14}$	2.41$_{\pm 0.13}$	1.99$_{\pm 0.12}$ (10%)	2.45 (25%)
rect-img	24.04$_{\pm 0.37}$	22.50$_{\pm 0.37}$	24.05$_{\pm 0.37}$	21.59$_{\pm 0.36}$ (25%)	23.00 (10%)
convex	19.13$_{\pm 0.34}$	18.63$_{\pm 0.34}$	18.41$_{\pm 0.34}$	19.06$_{\pm 0.34}$ (10%)	24.20 (10%)

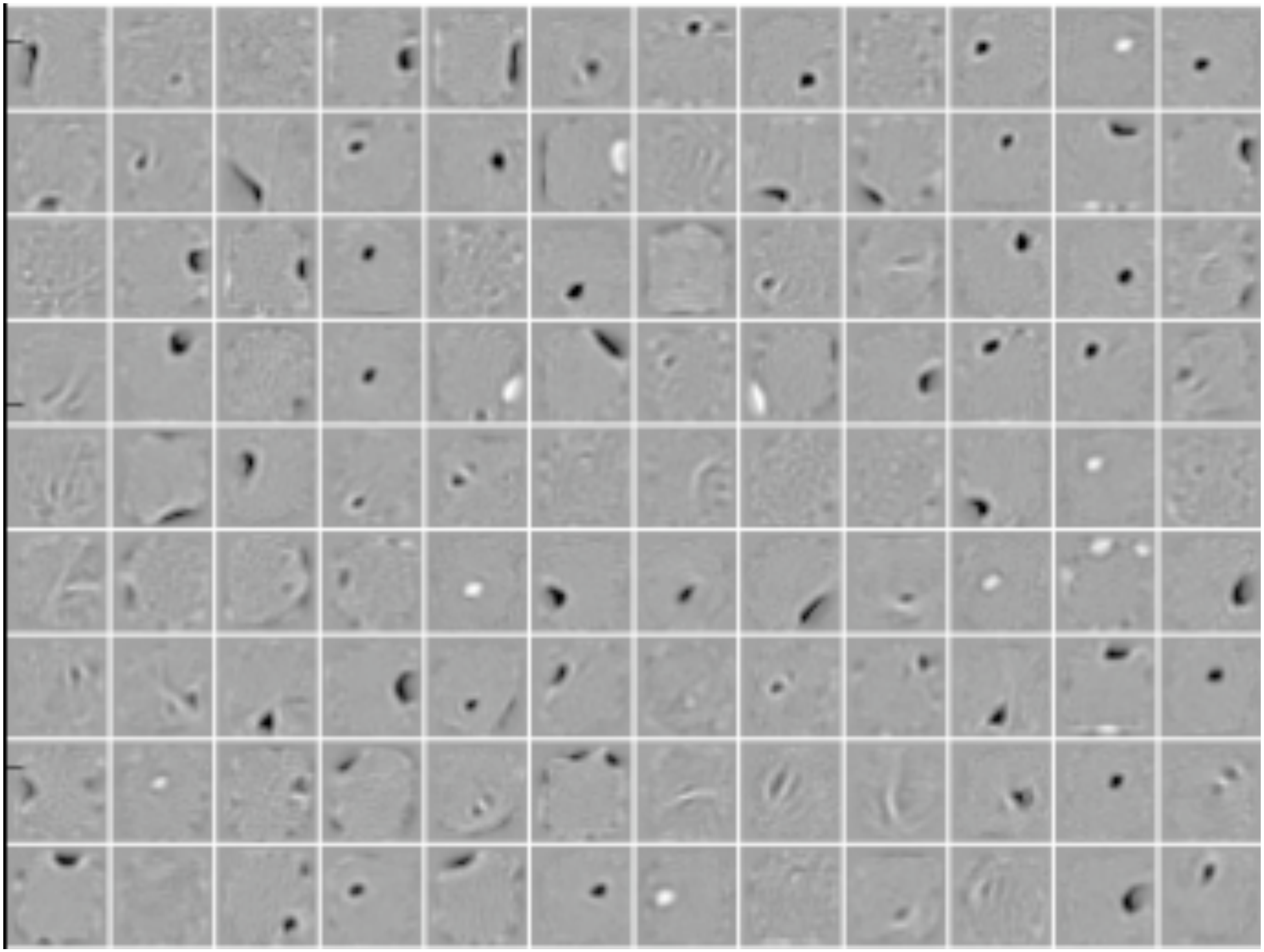
LEARNT FILTERS (0% DESTROYED)



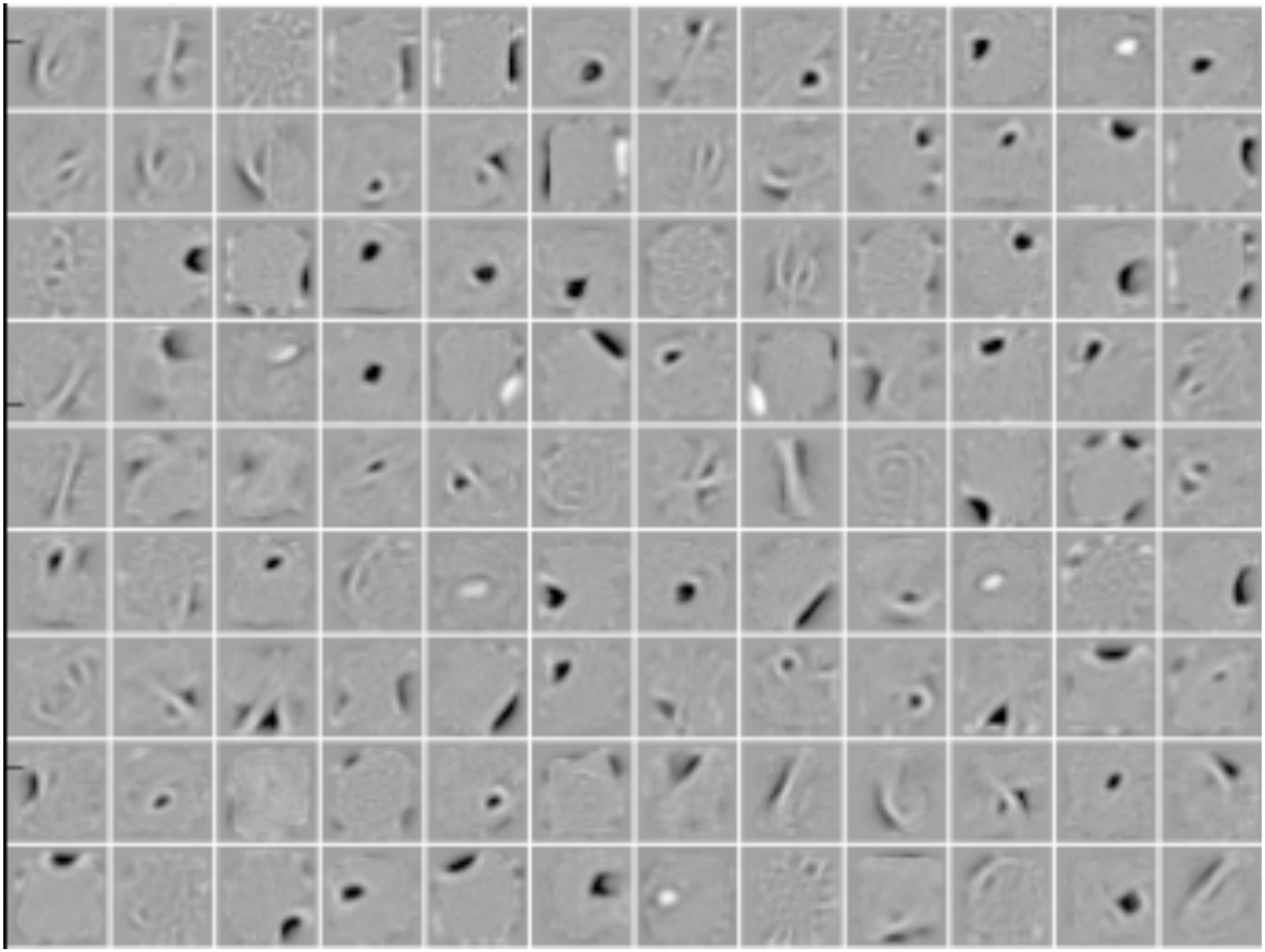
LEARNT FILTERS (10% DESTROYED)



LEARNT FILTERS (25% DESTROYED)



LEARNT FILTERS (50% DESTROYED)



CONCLUDING REMARKS

- Unsupervised initialization of layers with an explicit denoising criterion appears to help capture interesting structure in the input distribution.
- This leads to intermediate representations much better suited for subsequent learning tasks such as supervised classification.
- Resulting algorithm for learning deep networks is simple and improves on state-of-the-art on benchmark problems.
- Although our experimental focus was supervised classification, **SdA** is directly usable in a semi-supervised setting.
- We are currently investigating the effect of different types of corruption process, and applying the technique to recurrent nets.

READINGS

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007).
Greedy layer-wise training of deep networks.
In *NIPS 19*.

Gallinari, P., LeCun, Y., Thiria, S., and Fogelman-Soulie, F. (1987).
Memoires associatives distribuees.
In *Proceedings of COGNITIVA 87*, Paris, La Villette.

Hinton, G. E., Osindero, S., and Teh, Y. (2006).
A fast learning algorithm for deep belief nets.
Neural Computation, 18:1527–1554.

Hopfield, J. J. (1982).
Neural networks and physical systems with emergent collective computational abilities.
Proceedings of the National Academy of Sciences, USA, 79.

LeCun, Y. (1987).
Modèles connexionistes de l'apprentissage.
PhD thesis, Université de Paris VI.

READINGS

Ranzato, M., Poultney, C., Chopra, S., and LeCun, Y. (2007).
Efficient learning of sparse representations with an energy-based model.
In et al., J. P., editor, *Advances in Neural Information Processing Systems (NIPS 2006)*. MIT Press.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986).
Learning representations by back-propagating errors.
Nature, 323:533–536.