Deep Learning with Denoising Autoencoders

Pascal Vincent,
Hugo Larochelle, Yoshua Bengio, Pierre-Antoine Manzagol

Université de Montréal, LISA Lab

2008-03-25

The problem

- 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.
- Learning the parameters of deep architectures proved to be challenging!

Training deep architectures: attempted solutions

- **Solution 1**: initialize at random, and do gradient descent (Rumelhart et al., 1986).
  $\rightarrow$ disappointing performance. Stuck in poor solutions.
- **Solution 2**: Deep Belief Nets (Hinton et al., 2006): initialize by stacking Restricted Boltzmann Machines, fine-tune with Up-Down.
  $\rightarrow$ impressive performance.

  Key seems to be good unsupervised layer-by-layer initialization. . .
- **Solution 3**: initialize by stacking autoencoders, fine-tune with gradient descent. (Bengio et al., 2007; Ranzato et al., 2007)
  $\rightarrow$ Simple generic procedure, no sampling required.
  Performance almost as good as Solution 2

. . . but not quite. Can we do better?

Training deep architectures: attempted solutions

- **Solution 1**: initialize at random, and do gradient descent (Rumelhart et al., 1986).
  → disappointing performance. Stuck in poor solutions.
- **Solution 2**: Deep Belief Nets (Hinton et al., 2006): initialize by stacking Restricted Boltzmann Machines, fine-tune with Up-Down.
  → impressive performance.

  Key seems to be good unsupervised layer-by-layer initialization. . .
- **Solution 3**: initialize by stacking autoencoders, fine-tune with gradient descent. (Bengio et al., 2007; Ranzato et al., 2007)
  → Simple generic procedure, no sampling required.
  Performance almost as good as Solution 2

. . . but not quite. Can we do better?

Training deep architectures: attempted solutions

- **Solution 1**: initialize at random, and do gradient descent (Rumelhart et al., 1986).
  $\rightarrow$ disappointing performance. Stuck in poor solutions.

- **Solution 2**: Deep Belief Nets (Hinton et al., 2006): initialize by stacking Restricted Boltzmann Machines, fine-tune with Up-Down.
  $\rightarrow$ impressive performance.

  Key seems to be good unsupervised layer-by-layer initialization. . .

- **Solution 3**: initialize by stacking autoencoders, fine-tune with gradient descent. (Bengio et al., 2007; Ranzato et al., 2007)
  $\rightarrow$ Simple generic procedure, no sampling required.
  Performance almost as good as Solution 2

. . . but not quite. Can we do better?

Training deep architectures: attempted solutions

- **Solution 1**: initialize at random, and do gradient descent (Rumelhart et al., 1986).
  → disappointing performance. Stuck in poor solutions.
- **Solution 2**: Deep Belief Nets (Hinton et al., 2006): initialize by stacking Restricted Boltzmann Machines, fine-tune with Up-Down.
  → impressive performance.

  Key seems to be good unsupervised layer-by-layer initialization...
- **Solution 3**: initialize by stacking autoencoders, fine-tune with gradient descent. (Bengio et al., 2007; Ranzato et al., 2007)
  → Simple generic procedure, no sampling required.
  Performance almost as good as Solution 2

...but not quite. Can we do better?

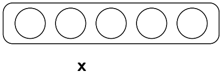
Can we do better?

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.
- $\rightarrow$ old notion of associative memory (motivated Hopfield models (Hopfield, 1982))

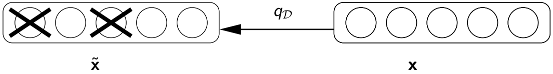
What we propose:
unsupervised initialization by explicit fill-in-the-blanks training.

The denoising autoencoder



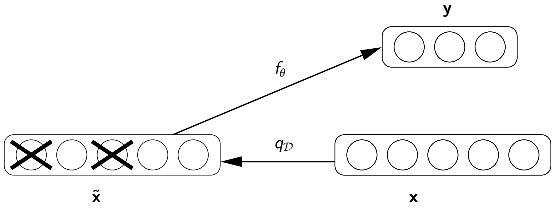
- Clean input $\mathbf{x} \in [0,1]^d$ is partially destroyed, yielding corrupted input: $\tilde{\mathbf{x}} \sim q_{\mathcal{D}}(\tilde{\mathbf{x}}|\mathbf{x})$.
- $\tilde{\mathbf{x}}$ is mapped to hidden representation $\mathbf{y} = f_\theta(\tilde{\mathbf{x}})$.
- From $\mathbf{y}$ we reconstruct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

The denoising autoencoder



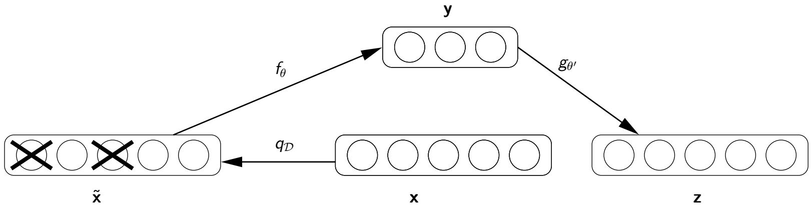
- Clean input $\mathbf{x} \in [0,1]^d$ is partially destroyed, yielding corrupted input: $\tilde{\mathbf{x}} \sim q_{\mathcal{D}}(\tilde{\mathbf{x}}|\mathbf{x})$.
- $\tilde{\mathbf{x}}$ is mapped to hidden representation $\mathbf{y} = f_\theta(\tilde{\mathbf{x}})$.
- From $\mathbf{y}$ we reconstruct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

The denoising autoencoder



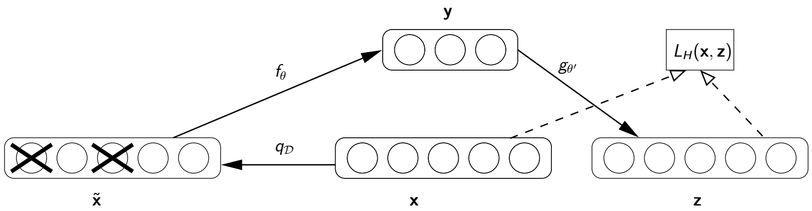
- Clean input $\mathbf{x} \in [0,1]^d$ is partially destroyed, yielding corrupted input: $\tilde{\mathbf{x}} \sim q_\mathcal{D}(\tilde{\mathbf{x}}|\mathbf{x})$.
- $\tilde{\mathbf{x}}$ is mapped to hidden representation $\mathbf{y} = f_\theta(\tilde{\mathbf{x}})$.
- From $\mathbf{y}$ we reconstruct a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

The denoising autoencoder



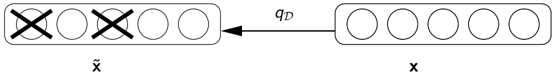
- Clean input $\mathbf{x} \in [0,1]^d$ is partially destroyed, yielding corrupted input: $\tilde{\mathbf{x}} \sim q_\mathcal{D}(\tilde{\mathbf{x}}|\mathbf{x})$.
- $\tilde{\mathbf{x}}$ is mapped to hidden representation $\mathbf{y} = f_\theta(\tilde{\mathbf{x}})$.
- From $\mathbf{y}$ we reconstruct a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

The 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"

The input corruption process $q_{\mathcal{D}}(\tilde{\mathbf{x}}|\mathbf{x})$



- 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 could be considered.

Form of parameterized mappings

We use standard sigmoid network layers:

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

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

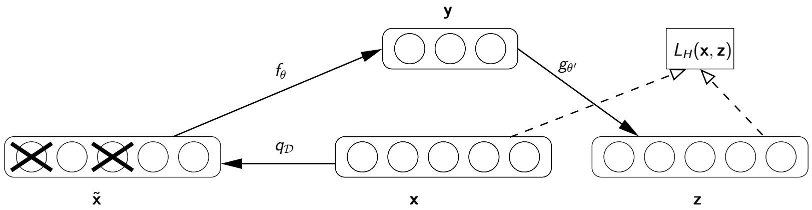
Form of parameterized mappings

We use standard sigmoid network layers:

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

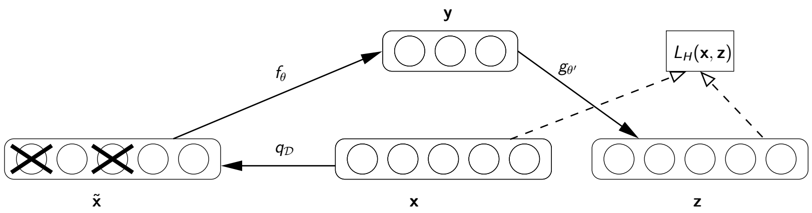
Denoising using autoencoders was actually introduced much earlier (LeCun, 1987; Gallinari et al., 1987), as an alternative to Hopfield networks (Hopfield, 1982).

Learning deep networks
Layer-wise initialization



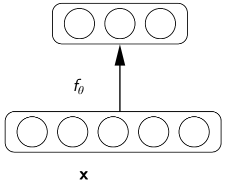
1. Learn first mapping f_θ by training as a denoising autoencoder.
2. Remove scaffolding. Use f_θ directly on input yielding higher level representation.
3. Learn next level mapping $f_\theta^{(2)}$ by training denoising autoencoder on current level representation.
4. Iterate to initialize subsequent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

Learning deep networks
Layer-wise initialization



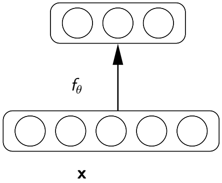
1. Learn first mapping f_θ by training as a denoising autoencoder.
2. Remove scaffolding. Use f_θ directly on input yielding higher level representation.
3. Learn next level mapping $f_\theta^{(2)}$ by training denoising autoencoder on current level representation.
4. Iterate to initialize subsequent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

Learning deep networks
Layer-wise initialization



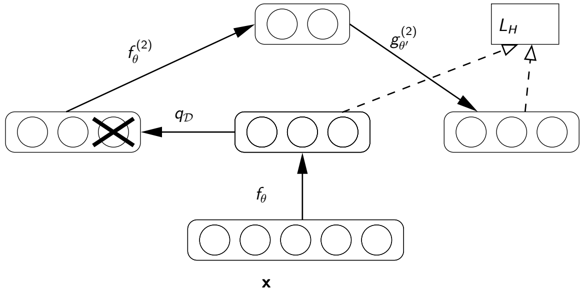
1. Learn first mapping f_θ by training as a denoising autoencoder.
2. Remove scaffolding. Use f_θ directly on input yielding higher level representation.
3. Learn next level mapping $f_\theta^{(2)}$ by training denoising autoencoder on current level representation.
4. Iterate to initialize subsequent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

Learning deep networks
Layer-wise initialization



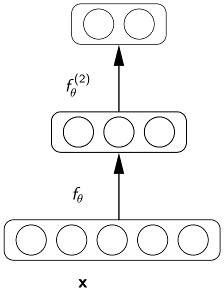
1. Learn first mapping f_θ by training as a denoising autoencoder.
2. Remove scaffolding. Use f_θ directly on input yielding higher level representation.
3. Learn next level mapping $f_\theta^{(2)}$ by training denoising autoencoder on current level representation.
4. Iterate to initialize subsequent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

Learning deep networks
Layer-wise initialization



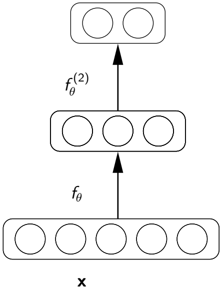
1. Learn first mapping f_θ by training as a denoising autoencoder.
2. Remove scaffolding. Use f_θ directly on input yielding higher level representation.
3. Learn next level mapping $f_\theta^{(2)}$ by training denoising autoencoder on current level representation.
4. Iterate to initialize subsequent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

Learning deep networks
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.

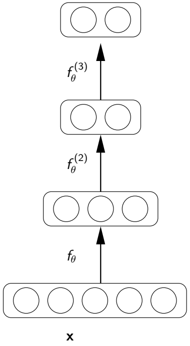
Learning deep networks
Layer-wise initialization



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.

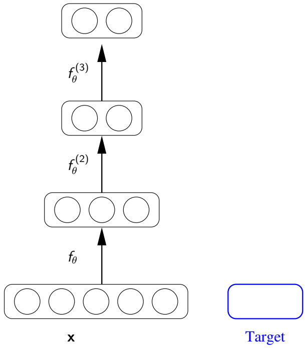
Learning deep networks
Supervised fine-tuning

- 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.



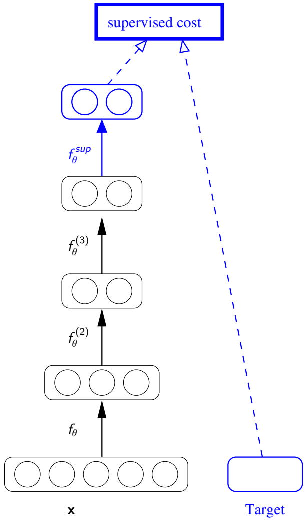
Learning deep networks
Supervised fine-tuning

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



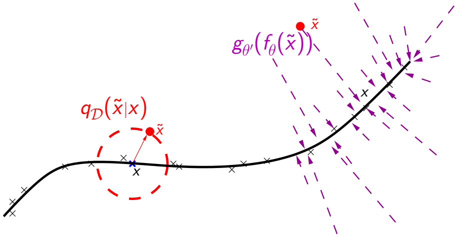
Learning deep networks
Supervised fine-tuning

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



Perspectives on denoising autoencoders

Manifold learning perspective



$q_{\mathcal{D}}(\tilde{x}|x)$
$\tilde{x}$
x
$g_{\theta'}(f_\theta(\tilde{x}))$
$\tilde{x}$
x

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.

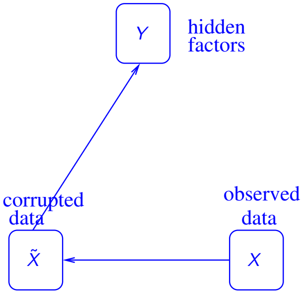
Perspectives on denoising autoencoders
Information theoretic perspective

- 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.*

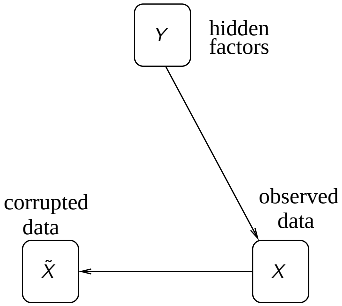
Perspectives on denoising autoencoders

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.



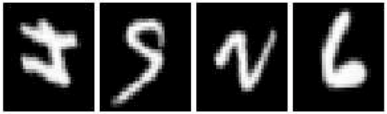
variational model



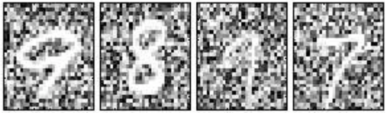
generative model

Benchmark problems
Variations on MNIST digit classification

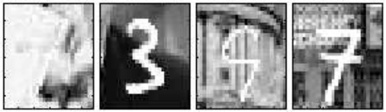
basic: subset of original MNIST digits: 10 000 training samples, 2 000 validation samples, 50 000 test samples.



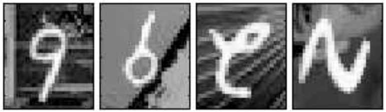
rot: applied random rotation (angle between 0 and 2π radians)



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



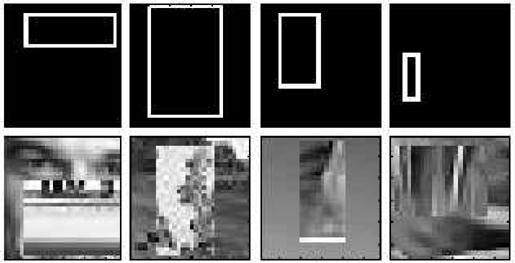
bg-img: background is random patch from one of 20 images



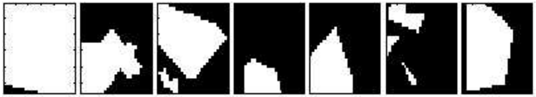
rot-bg-img: combination of rotation and background image

Benchmark problems
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.



Experiments

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
Results

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

Performance comparison
Results

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

Performance comparison

Results

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

Performance comparison
Results

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

Performance comparison

Results

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

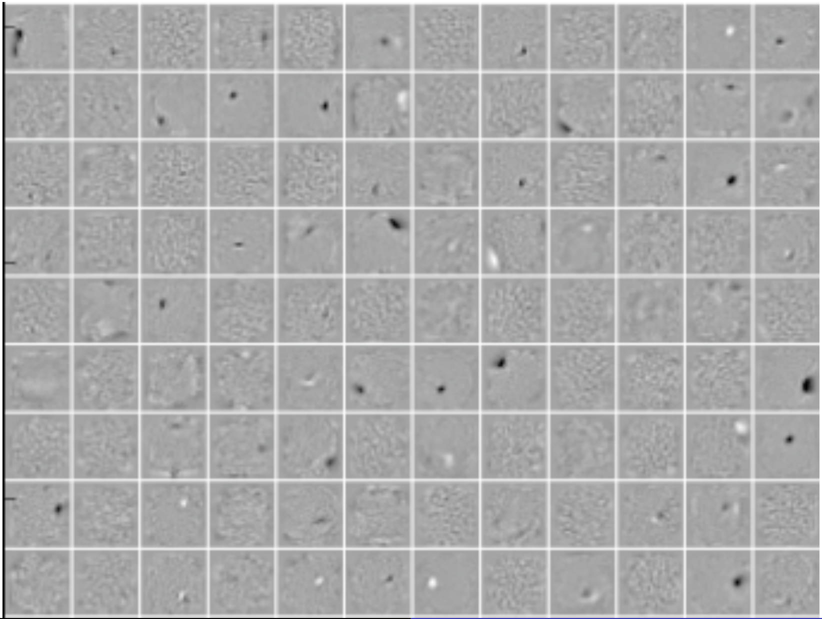
Performance comparison
Results

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

Performance comparison
Results

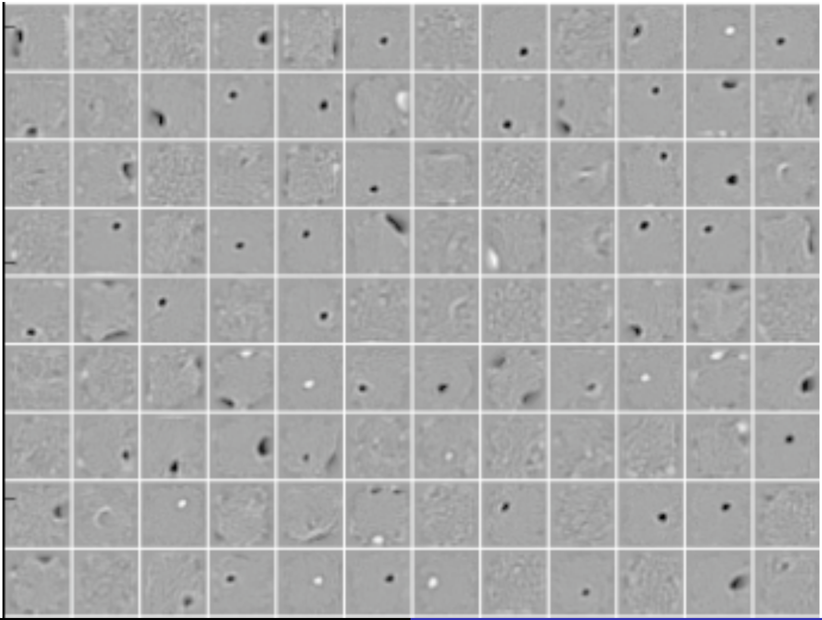
Dataset	SVM$_{rbf}$	DBN-3	SAA-3	SdA-3 (ν)
basic	$3.03_{\pm 0.15}$	$3.11_{\pm 0.15}$	$3.46_{\pm 0.16}$	$2.80_{\pm 0.14}$ (10%)
rot	$11.11_{\pm 0.28}$	$10.30_{\pm 0.27}$	$10.30_{\pm 0.27}$	$10.29_{\pm 0.27}$ (10%)
bg-rand	$14.58_{\pm 0.31}$	$6.73_{\pm 0.22}$	$11.28_{\pm 0.28}$	$10.38_{\pm 0.27}$ (40%)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

Learnt filters
0 % destroyed



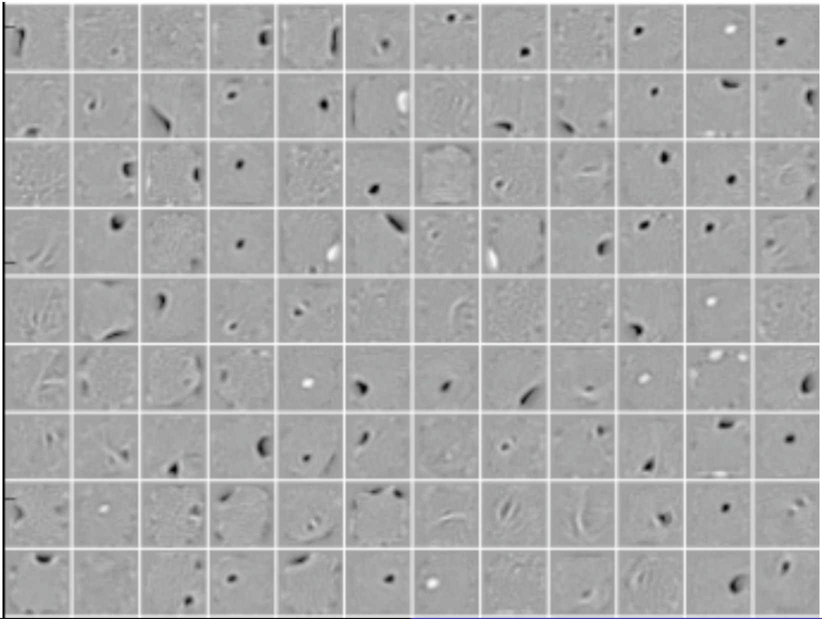
Learnt filters

10 % destroyed



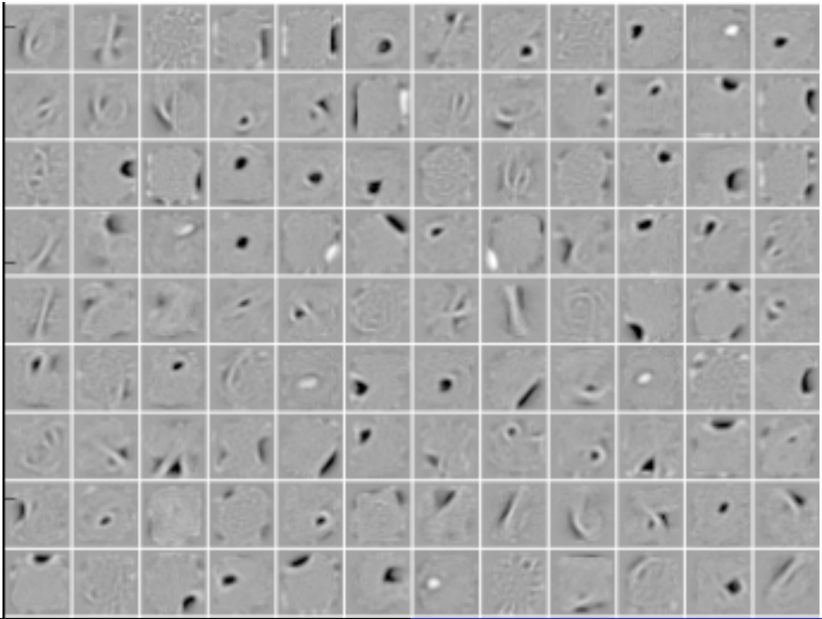
Learnt filters

25 % destroyed



Learnt filters

50 % destroyed



Conclusion and future work

- 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.
- Future work will investigate the effect of different types of corruption process.

THANK YOU!

Performance comparison

Dataset	SVM$_{rbf}$	SVM$_{poly}$	DBN-1	DBN-3	SAA-3	SdA-3 (ν)
basic	3.03$\pm$0.15	3.69$\pm$0.17	3.94$\pm$0.17	3.11$\pm$0.15	3.46$\pm$0.16	2.80$\pm$0.14 (10%)
rot	11.11$\pm$0.28	15.42$\pm$0.32	14.69$\pm$0.31	10.30$\pm$0.27	10.30$\pm$0.27	10.29$\pm$0.27 (10%)
bg-rand	14.58$\pm$0.31	16.62$\pm$0.33	9.80$\pm$0.26	6.73$\pm$0.22	11.28$\pm$0.28	10.38$\pm$0.27 (40%)
bg-img	22.61$\pm$0.37	24.01$\pm$0.37	16.15$\pm$0.32	16.31$\pm$0.32	23.00$\pm$0.37	16.68$\pm$0.33 (25%)
rot-bg-img	55.18$\pm$0.44	56.41$\pm$0.43	52.21$\pm$0.44	47.39$\pm$0.44	51.93$\pm$0.44	44.49$\pm$0.44 (25%)
rect	2.15$\pm$0.13	2.15$\pm$0.13	4.71$\pm$0.19	2.60$\pm$0.14	2.41$\pm$0.13	1.99$\pm$0.12 (10%)
rect-img	24.04$\pm$0.37	24.05$\pm$0.37	23.69$\pm$0.37	22.50$\pm$0.37	24.05$\pm$0.37	21.59$\pm$0.36 (25%)
convex	19.13$\pm$0.34	19.82$\pm$0.35	19.92$\pm$0.35	18.63$\pm$0.34	18.41$\pm$0.34	19.06$\pm$0.34 (10%)

red when confidence intervals overlap.

References

Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2007). Greedy layer-wise training of deep networks. *Advances in Neural Information Processing Systems 19* (pp. 153–160). MIT Press.

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

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

Hopfield, 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*. Doctoral dissertation, Université de Paris VI.

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

Rumelhart, D., Hinton, G., & Williams, R. (1986). Learning representations by back-propagating errors. *Nature*, *323*, 533–536.