Deep Learning with Denoising Autoencoders

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

Université de Montréal, LISA Lab

2008-03-25
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!
**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?
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?
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?
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 $x \in [0, 1]^d$ is partially destroyed, yielding corrupted input: $\tilde{x} \sim q_D(\tilde{x}|x)$.
- $\tilde{x}$ is mapped to hidden representation $y = f_\theta(\tilde{x})$.
- From $y$ we reconstruct $z = g_{\theta'}(y)$.
- 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: $\tilde{\mathbf{x}} \sim q_D(\tilde{\mathbf{x}}|\mathbf{x})$.

- $\tilde{\mathbf{x}}$ 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: $\tilde{\mathbf{x}} \sim q_D(\tilde{\mathbf{x}}|\mathbf{x})$.

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

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

- Choose a fixed proportion \( \nu \) of components of \( x \) 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:

- \( y = f_\theta(\tilde{x}) = \text{sigmoid}(W \tilde{x} + b) \)
  - \( d' \times d \) \( d' \times 1 \)

- \( g_{\theta'}(y) = \text{sigmoid}(W' y + b') \)
  - \( d \times d' \) \( d \times 1 \)

Denoising using autoencoders was actually introduced much earlier (LeCun, 1987; Gallinari et al., 1987), as an alternative to Hopfield networks (Hopfield, 1982).
We use standard sigmoid network layers:

- \( y = f_\theta(\tilde{x}) = \text{sigmoid}(W \tilde{x} + b) \)

- \( g_{\theta'}(y) = \text{sigmoid}(W' y + b') \).

Denoising using autoencoders was actually introduced much earlier (LeCun, 1987; Gallinari et al., 1987), as an alternative to Hopfield networks (Hopfield, 1982).
Learn first mapping $f_\theta$ by training as a denoising autoencoder.

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

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

Iterate to initialize subsequent layers.
Learn first mapping $f_\theta$ by training as a denoising autoencoder.

2. Remove scaffolding. Use $f_\theta$ 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.
Learn first mapping $f_\theta$ by training as a denoising autoencoder.

2. Remove scaffolding. Use $f_\theta$ 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.
1. Learn first mapping $f_\theta$ by training as a denoising autoencoder.
2. Remove scaffolding. Use $f_\theta$ directly on input yielding higher level representation.
3. Learn next level mapping $f^{(2)}_\theta$ by training denoising autoencoder on current level representation.
4. Iterate to initialize subsequent layers.
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.
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.
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.
Denoising autoencoder can be seen as a way to learn a manifold:

- Suppose training data (×) concentrate near a low-dimensional manifold.
- Corrupted examples (●) are obtained by applying corruption process $q_D(\tilde{X}|X)$ and will lie farther from the manifold.
- The model learns with $p(X|\tilde{X})$ to “project them back” onto the manifold.
- Intermediate representation $Y$ can be interpreted as a coordinate system for points on the manifold.
Consider $X \sim q(X)$, $q$ unknown. $\tilde{X} \sim q_D(\tilde{X}|X)$. $Y = f_\theta(\tilde{X})$.

It can be shown that minimizing the expected reconstruction error amounts to maximizing a lower bound on mutual information $I(X; Y)$.

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.
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.
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 $2\pi$ radians)

**bg-rand**: background made of random pixels (value in 0...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.
We compared the following algorithms on the benchmark problems:

- **SVM\textsubscript{rbf}**: Support 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 classification performance on validation set. (In particular hidden-layer sizes, and $\nu$ for **SdA-3**).
Performance comparison

Results

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

### Results

<table>
<thead>
<tr>
<th>Dataset</th>
<th>SVM&lt;sub&gt;rbf&lt;/sub&gt;</th>
<th>DBN-3</th>
<th>SAA-3</th>
<th>SdA-3 (ν)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>basic</strong></td>
<td>3.03±0.15</td>
<td>3.11±0.15</td>
<td>3.46±0.16</td>
<td>2.80±0.14 (10%)</td>
</tr>
<tr>
<td><strong>rot</strong></td>
<td>11.11±0.28</td>
<td>10.30±0.27</td>
<td>10.30±0.27</td>
<td>10.29±0.27 (10%)</td>
</tr>
<tr>
<td><strong>bg-rand</strong></td>
<td>14.58±0.31</td>
<td>6.73±0.22</td>
<td>11.28±0.28</td>
<td>10.38±0.27 (40%)</td>
</tr>
<tr>
<td><strong>bg-img</strong></td>
<td>22.61±0.37</td>
<td>16.31±0.32</td>
<td>23.00±0.37</td>
<td>16.68±0.33 (25%)</td>
</tr>
<tr>
<td><strong>rot-bg-img</strong></td>
<td>55.18±0.44</td>
<td>47.39±0.44</td>
<td>51.93±0.44</td>
<td>44.49±0.44 (25%)</td>
</tr>
<tr>
<td><strong>rect</strong></td>
<td>2.15±0.13</td>
<td>2.60±0.14</td>
<td>2.41±0.13</td>
<td>1.99±0.12 (10%)</td>
</tr>
<tr>
<td><strong>rect-img</strong></td>
<td>24.04±0.37</td>
<td>22.50±0.37</td>
<td>24.05±0.37</td>
<td>21.59±0.36 (25%)</td>
</tr>
<tr>
<td><strong>convex</strong></td>
<td>19.13±0.34</td>
<td>18.63±0.34</td>
<td>18.41±0.34</td>
<td>19.06±0.34 (10%)</td>
</tr>
</tbody>
</table>
## Performance comparison

### Results

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$\text{SVM}_{rbf}$</th>
<th>DBN-3</th>
<th>SAA-3</th>
<th>SdA-3 ($\nu$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic</td>
<td>3.03 ± 0.15</td>
<td>3.11 ± 0.15</td>
<td>3.46 ± 0.16</td>
<td>2.80 ± 0.14 (10%)</td>
</tr>
<tr>
<td>rot</td>
<td>11.11 ± 0.28</td>
<td>10.30 ± 0.27</td>
<td>10.30 ± 0.27</td>
<td>10.29 ± 0.27 (10%)</td>
</tr>
<tr>
<td>bg-rand</td>
<td>14.58 ± 0.31</td>
<td>6.73 ± 0.32</td>
<td>11.28 ± 0.28</td>
<td>10.38 ± 0.27 (40%)</td>
</tr>
<tr>
<td>bg-img</td>
<td>22.61 ± 0.37</td>
<td>16.31 ± 0.32</td>
<td>23.00 ± 0.37</td>
<td>16.68 ± 0.33 (25%)</td>
</tr>
<tr>
<td>rot-bg-img</td>
<td>55.18 ± 0.44</td>
<td>47.39 ± 0.44</td>
<td>51.93 ± 0.44</td>
<td>44.49 ± 0.44 (25%)</td>
</tr>
<tr>
<td>rect</td>
<td>2.15 ± 0.13</td>
<td>2.60 ± 0.14</td>
<td>2.41 ± 0.13</td>
<td>1.99 ± 0.12 (10%)</td>
</tr>
<tr>
<td>rect-img</td>
<td>24.04 ± 0.37</td>
<td>22.50 ± 0.37</td>
<td>24.05 ± 0.37</td>
<td>21.59 ± 0.36 (25%)</td>
</tr>
<tr>
<td>convex</td>
<td>19.13 ± 0.34</td>
<td>18.63 ± 0.34</td>
<td>18.41 ± 0.34</td>
<td>19.06 ± 0.34 (10%)</td>
</tr>
</tbody>
</table>
## Performance comparison

### Results

<table>
<thead>
<tr>
<th>Dataset</th>
<th>SVM(_{rbf})</th>
<th>DBN-3</th>
<th>SAA-3</th>
<th>SdA-3 ((\nu))</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic</td>
<td>3.03±0.15</td>
<td>3.11±0.15</td>
<td>3.46±0.16</td>
<td>2.80±0.14 (10%)</td>
</tr>
<tr>
<td>rot</td>
<td>11.11±0.28</td>
<td>10.30±0.27</td>
<td>10.30±0.27</td>
<td>10.29±0.27 (10%)</td>
</tr>
<tr>
<td>bg-rand</td>
<td>14.58±0.31</td>
<td>6.73±0.23</td>
<td>11.28±0.28</td>
<td>10.38±0.27 (40%)</td>
</tr>
<tr>
<td>bg-img</td>
<td>22.61±0.37</td>
<td>16.31±0.32</td>
<td>23.00±0.37</td>
<td>16.68±0.33 (25%)</td>
</tr>
<tr>
<td>rot-bg-img</td>
<td>55.18±0.44</td>
<td>47.39±0.44</td>
<td>51.93±0.44</td>
<td>44.49±0.44 (25%)</td>
</tr>
<tr>
<td>rect</td>
<td>2.15±0.13</td>
<td>2.60±0.14</td>
<td>2.41±0.13</td>
<td>1.99±0.12 (10%)</td>
</tr>
<tr>
<td>rect-img</td>
<td>24.04±0.37</td>
<td>22.50±0.37</td>
<td>24.05±0.37</td>
<td>21.59±0.36 (25%)</td>
</tr>
<tr>
<td>convex</td>
<td>19.13±0.34</td>
<td>18.63±0.34</td>
<td>18.41±0.34</td>
<td>19.06±0.34 (10%)</td>
</tr>
</tbody>
</table>
## Performance comparison

### Results

<table>
<thead>
<tr>
<th>Dataset</th>
<th>SVM(_{rbf})</th>
<th>DBN-3</th>
<th>SAA-3</th>
<th>SdA-3 ((\nu))</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic</td>
<td>3.03 ± 0.15</td>
<td>3.11 ± 0.15</td>
<td>3.46 ± 0.16</td>
<td>2.80 ± 0.14 (10%)</td>
</tr>
<tr>
<td>rot</td>
<td>11.11 ± 0.28</td>
<td>10.30 ± 0.27</td>
<td>10.30 ± 0.27</td>
<td>10.29 ± 0.27 (10%)</td>
</tr>
<tr>
<td>bg-rand</td>
<td>14.58 ± 0.31</td>
<td>6.73 ± 0.22</td>
<td>11.28 ± 0.26</td>
<td>10.38 ± 0.27 (40%)</td>
</tr>
<tr>
<td>bg-img</td>
<td>22.61 ± 0.37</td>
<td>16.31 ± 0.32</td>
<td>23.00 ± 0.37</td>
<td>16.68 ± 0.33 (25%)</td>
</tr>
<tr>
<td>rot-bg-img</td>
<td>55.18 ± 0.44</td>
<td>47.39 ± 0.44</td>
<td>51.93 ± 0.44</td>
<td>44.49 ± 0.44 (25%)</td>
</tr>
<tr>
<td>rect</td>
<td>2.15 ± 0.13</td>
<td>2.60 ± 0.14</td>
<td>2.41 ± 0.13</td>
<td>1.99 ± 0.12 (10%)</td>
</tr>
<tr>
<td>rect-img</td>
<td>24.04 ± 0.37</td>
<td>22.50 ± 0.37</td>
<td>24.05 ± 0.37</td>
<td>21.59 ± 0.35 (25%)</td>
</tr>
<tr>
<td>convex</td>
<td>19.13 ± 0.34</td>
<td>18.63 ± 0.34</td>
<td>18.41 ± 0.34</td>
<td>19.06 ± 0.34 (10%)</td>
</tr>
</tbody>
</table>
## Performance comparison

### Results

<table>
<thead>
<tr>
<th>Dataset</th>
<th>SVM$_{rbf}$</th>
<th>DBN-3</th>
<th>SAA-3</th>
<th>SdA-3 ($\nu$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic</td>
<td>3.03 ± 0.15</td>
<td>3.11 ± 0.15</td>
<td>3.46 ± 0.16</td>
<td>2.80 ± 0.14 (10%)</td>
</tr>
<tr>
<td>rot</td>
<td>11.11 ± 0.28</td>
<td>10.30 ± 0.27</td>
<td>10.30 ± 0.27</td>
<td>10.29 ± 0.27 (10%)</td>
</tr>
<tr>
<td>bg-rand</td>
<td>14.58 ± 0.31</td>
<td>6.73 ± 0.22</td>
<td>11.28 ± 0.28</td>
<td>10.38 ± 0.27 (40%)</td>
</tr>
<tr>
<td>bg-img</td>
<td>22.61 ± 0.37</td>
<td>16.31 ± 0.32</td>
<td>23.00 ± 0.37</td>
<td>16.68 ± 0.33 (25%)</td>
</tr>
<tr>
<td>rot-bg-img</td>
<td>55.18 ± 0.44</td>
<td>47.39 ± 0.44</td>
<td>51.93 ± 0.44</td>
<td>44.49 ± 0.44 (25%)</td>
</tr>
<tr>
<td>rect</td>
<td>2.15 ± 0.13</td>
<td>2.60 ± 0.14</td>
<td>2.41 ± 0.13</td>
<td>1.99 ± 0.12 (10%)</td>
</tr>
<tr>
<td>rect-img</td>
<td>24.04 ± 0.37</td>
<td>22.50 ± 0.37</td>
<td>24.05 ± 0.37</td>
<td>21.59 ± 0.35 (25%)</td>
</tr>
<tr>
<td>convex</td>
<td>19.13 ± 0.34</td>
<td>18.63 ± 0.34</td>
<td>18.41 ± 0.34</td>
<td>19.06 ± 0.34 (10%)</td>
</tr>
</tbody>
</table>
### Performance comparison

#### Results

| Dataset  | \( \text{SVM}_{rbf} \)  | DBN-3     | SAA-3     | SdA-3 \((\nu)\)     |
|----------|----------------|-----------|-----------|----------------|-----------------|
| basic    | 3.03±0.15      | 3.11±0.15 | 3.46±0.16 | 2.80±0.14 \(10\%\) |
| rot      | 11.11±0.28     | 10.30±0.27 | 10.30±0.27 | 10.29±0.27 \(10\%\) |
| bg-rand  | 14.58±0.31     | 6.73±0.22 | 11.28±0.28 | 10.38±0.27 \(40\%\) |
| bg-img   | 22.61±0.37     | 16.31±0.32 | 23.00±0.37 | 16.68±0.33 \(25\%\) |
| rot-bg-img | 55.18±0.44 | 47.39±0.44 | 51.93±0.44 | 44.49±0.44 \(25\%\) |
| rect     | 2.15±0.13      | 2.60±0.14 | 2.41±0.13 | 1.99±0.12 \(10\%\) |
| rect-img | 24.04±0.37     | 22.50±0.37 | 24.05±0.37 | 21.59±0.36 \(25\%\) |
| convex   | 19.13±0.34     | 18.63±0.34 | 18.41±0.34 | 19.06±0.34 \(10\%\) |
Learnt filters
0 % destroyed
Learnt filters
25% destroyed
Learnt filters
50 % destroyed
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

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

*red* when confidence intervals overlap.
References


